ANDRÉIA SIAS RODRIGUES

UMA PROPOSTA DIFERENCIADA DE TAXONOMIA PARA MECANISMOS DE CONTROLE DE CONCORRÊNCIA DE BANCOS DE DADOS EM AMBIENTES SEM FIO

FLORIANÓPOLIS – SC
2004

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CURSO DE PÓS-GRADUAÇÃO
EM CIÊNCIA DA COMPUTAÇÃO

ANDRÉIA SIAS RODRIGUES

UMA PROPOSTA DIFERENCIADA DE TAXONOMIA PARA MECANISMOS DE CONTROLE DE CONCORRÊNCIA DE BANCOS DE DADOS EM AMBIENTES SEM FIO

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Orientador:
MURILO SILVA DE CAMARGO

Florianópolis, Fevereiro de 2004

UMA PROPOSTA DIFERENCIADA DE TAXONOMIA PARA MECANISMOS DE CONTROLE DE CONCORRÊNCIA DE BANCOS DE DADOS EM AMBIENTES SEM FIO

Andréia Sias Rodrigues

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação (Sistema de Computação) e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____
Raul Sidnei Wazlawick
(Coordenador do Curso)

Banca Examinadora

_____
Prof. Dr. Murilo Silva de Camargo
(Orientador e Presidente da Banca)

_____
Prof. Dr. Mário Antonio Ribeiro Dantas

_____
Prof. Dr. Rosvelter Coelho da Costa

_____
Prof. Dr. Vitório Bruno Mazzola

Aos meus pais Carlos e Mara,
pela compreensão e carinho.

À minha irmã Marcela pelo companheirismo.

À minha avó materna Selma pelo incentivo, carinho e atenção.

Agradeço a Deus pela saúde, pelas oportunidades e por ter traçado caminhos tão cheio de alegrias.

Agradeço à minha família, pelo apoio incondicional, pelo conforto em momentos difíceis e por se mostrarem sempre otimistas e confiantes, e, apesar da distância, presentes.

Agradeço aos amigos de trabalho, aos amigos do mestrado, e tantos outros por terem me acompanhado e me dado força nessa jornada.

Agradeço a toda a equipe do Núcleo de Processamento de Dados, onde estagiei, principalmente ao Edson Melo, que permitiram a realização deste trabalho, através da utilização de ambiente e recursos disponíveis.

Aos professores integrantes do CPGCC pelo aprendizado e principalmente ao Professor Mário Dantas, pelo incentivo e sugestões ao texto.

E um agradecimento especial ao Professor e amigo Murilo, pela oportunidade e principalmente pela dedicação.

UMA PROPOSTA DIFERENCIADA DE TAXONOMIA PARA MECANISMOS DE CONTROLE DE CONCORRÊNCIA DE BANCOS DE DADOS EM AMBIENTES SEM FIO

Andréia Sias Rodrigues

Fevereiro / 2004

Orientador: Murilo Silva de Camargo.

Área de Conhecimento: Sistemas de Computação.

Palavras-chave: Banco de Dados Móveis, Taxonomia e Controle de Concorrência.

Número de Páginas: 135.

RESUMO:

O objetivo fundamental do controle de concorrência em banco de dados é assegurar que a execução concorrente de transações não resulte na perda da consistência do banco de dados, ou seja, é necessário assegurar o isolamento das transações. No que diz respeito aos bancos de dados móveis, os mecanismos de controle de concorrência aplicados em bancos de dados tradicionais, ou até mesmo distribuídos, não satisfazem as restrições impostas pelo ambiente de computação móvel, como mobilidade das unidades, as freqüentes desconexões de rede, a baixa largura de banda e a portabilidade.

Baseando-se na referida fundamentação, neste trabalho é feito um estudo bibliográfico dos principais modelos de transações móveis, evidenciando suas arquiteturas, modos de processamento, tipos de transações utilizadas, traçando um comparativo de como é feito o suporte das propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) em cada modelo estudado. Com estas informações é feita uma análise dos mecanismos de controle de concorrência utilizados em cada modelo.

De acordo com as necessidades dos modelos de transações investigados na literatura, a taxonomia proposta tem como diferencial a apresentação da abordagem híbrida, onde os modelos de transações móveis poderão obter um melhor desempenho utilizando o modo pessimista, quando a conectado ao banco de dados, e otimista quando desconectado.

A DIFFERENTIATED PROPOSAL OF A TAXONOMY TO CONCURRENCY CONTROL MECHANISMS FOR DATABASES IN ENVIRONMENTS WIRELESS

Andréia Sias Rodrigues

Fevereiro / 2004

Advisor: Murilo Silva de Camargo.

Area of Concentration: Computing Systems.

Keywords: Mobile Databases, Taxonomy, Concurrency Control.

Number of Pages: 135.

ABSTRACT:

The main objective of the concurrency control in database is to ensure that the concurrency transactions execution does not result in loss of the database consistency, it means, that is necessary to ensure the transactions isolation. At respect about mobile databases, the concurrency control mechanisms applied in traditional databases, or even though distributed, do not satisfy the constraints imposed by the mobile computation environment, such as unit mobility, the frequent disconnections, low bandwidth and the portability.

Based in these arguments, this work presents a bibliographical study of the main transactions mobile models, evidencing its architectures, processing methods, transactions types, tracing a comparative study about how the support of properties ACID in each studied model is made.

In accordance with the necessities of the transactions mobile models investigated in literature, the taxonomy proposal has as distinguishing the presentation of the hybrid method, where the models of mobile transactions will be able to get one better performance using the pessimistic way, when the connected one to the data base, and optimist when detached

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

Constantes avanços em tecnologias de redes de computadores e de dispositivos de computação portáteis possibilitaram que usuários pudessem acessar informações através de uma conexão sem fio (*wireless*) e se movimentarem preservando esta conexão. A competitividade entre empresas e a busca incessante pela informação vêm proporcionando um investimento cada vez maior em pesquisas nesta área. A necessidade de transmissão e consultas a dados em tempo real, independente de horário e local, se tornou imprescindível.. Estamos entrando em uma nova era para a computação: a era da *computação móvel*.

Com a introdução da computação móvel, as diversas áreas computacionais tiveram que acompanhar esta evolução. Em se tratando de buscas de informações, a área de banco de dados têm sido uma das mais exploradas, visto que consultas aos grandes ou até mesmo pequenos bancos de dados se tornaram parte fundamental deste crescimento.

A computação móvel apesar de atraente traz consigo uma série de obstáculos [FORMAN, 1994], [IMIELINSKI & BADRINATH, 1994] e [CLARK & DEMIR, 2003], por exemplo, a baixa largura de banda, o alto custo de comunicação em redes sem fio, a maior susceptibilidade a falhas de comunicação e a autonomia das baterias dos dispositivos portáteis.

Devido a estas intrínsecas limitações impostas pelo ambiente de computação móvel, as transações precisaram suportar estas complicações e conseqüentemente passaram a ter outras características e as propriedades ACID não puderam mais ser exigidas com rigidez. Novos modelos de transações foram propostos com o objetivo de tratar esses problemas mantendo sempre a consistência do banco de dados.

Para assegurar a execução isolada de transações e melhorar a concorrência, um aspecto fundamental é o mecanismo de controle de concorrência utilizado. Com base nesta fundamentação, este trabalho apresenta um estudo bibliográfico dos modelos de

transações móveis, evidenciando suas arquiteturas, modos de processamento e traçando um comparativo de como é feito o suporte das propriedades ACID em cada modelo estudado e ainda apresenta as vantagens e desvantagens dos mecanismos de controle de concorrência estudados.

Além disso, com base nos mecanismos de controle de concorrência utilizados nos modelos de transações móveis abordados e com base em técnicas utilizadas para manter o isolamento das transações no ambiente móvel, é feita uma proposta de taxonomia de mecanismos de controle de concorrência para banco de dados móveis, visando classificar os algoritmos de acordo com o suporte às complicações impostas pelo ambiente móvel.

Entre as complicações abordadas, um aspecto fundamental é a mobilidade dos clientes móveis, que hora podem estar conectados e hora desconectados. Outros aspectos importantes são a largura de banda para transmissão de dados e o poder de processamento menor do que de unidades fixas. As transações devem ser tratadas de forma que não sejam abortadas devido a sua longa duração e desconexões da unidade móvel não sejam consideradas como falhas.

O restante desta dissertação está organizado da seguinte forma:

O Capítulo 2 apresenta uma visão geral a respeito das questões que envolvem o paradigma de Bancos de Dados *Wireless*, bem como conceitos, arquiteturas abordadas, e outras questões relacionadas a este ambiente que servirão como base para todo o trabalho.

O Capítulo 3 aborda em detalhes os modelos de transações móveis, suas características, modos de processamento, aplicações e suas restrições. Além disso, apresenta tabelas comparativas entre os modelos, analisando as propostas de cada modelo para fornecer o suporte às propriedades ACID.

O Capítulo 4 traz uma proposta de taxonomia de controle de concorrência para estes bancos de dados de acordo com suas primitivas de sincronização, com base em tecnologias de controle de concorrência utilizadas nos modelos de transações móveis e também em ambiente de broadcast de dados no ambiente móvel. Proporcionando uma classificação dos algoritmos estudados e apresentados neste trabalho.

Finalmente, o Capítulo 5 apresenta as conclusões sobre o trabalho de pesquisa efetuado e as propostas para trabalhos futuros.

2 Banco de Dados *Wireless*

Neste capítulo serão abordados dois grandes conceitos de banco de dados sem fio *wireless*, que servirão como base para todo o trabalho: o Gerenciamento de Transações e o Controle de Concorrência. Outros conceitos pertinentes a este ambiente, como Processamento de Consultas, Replicação de Dados, Recuperação de Falhas e Segurança não serão abordados.

O Gerenciamento de Transações (GT) envolve os conceitos de transações móveis e as formas adequadas para gerenciar estas transações em um ambiente móvel. Uma transação móvel é um tipo de transação distribuída, onde algumas partes do processamento são executadas em uma unidade móvel e outras partes nas unidades fixas [DUNHAM & KUMAR, 1998].

O grande objetivo do Controle de Concorrência é assegurar que a execução concorrente de transações não resulte em perda da consistência do banco de dados. Para atingir este objetivo são utilizados vários mecanismos que serão estudados no decorrer do trabalho.

O surgimento das redes móveis possibilitou a troca de informações independente de localização física. Esta tecnologia permitiu que a necessidade de acesso remoto a uma base de dados *anytime-anywhere*. A próxima sessão apresenta os tipos de redes móveis.

2.1 Tipos de Redes Móveis

Existem dois modelos possíveis para o ambiente móvel: as **redes móveis com infra-estrutura** e as **redes com estruturas temporais** (redes *ad hoc*). Nas redes *ad hoc,* uma estação móvel pode comunicar-se diretamente com outra dentro do seu raio de atuação, ou pode usar uma terceira como repetidora do sinal. Nas redes com infra-estrutura as unidades móveis comunicam-se através de uma estação de suporte à mobilidade.

Os principais termos utilizados em computação móvel de acordo com [ROCHA, 2001] [TEWARI & GRILLO, 1995] são:

- Rede Fixa: rede formada por elementos convencionais ou fixos conectados fisicamente;
- Estação Fixa: estação conectada fisicamente à rede. Esta estação possui localização geográfica e endereço de rede fixo;
- Unidade ou estação móvel móvel, UM: Estação que não possui uma localização geográfica e não tem endereço físico fixo;
- MSS ou Estação base (*Mobile Station Support*): Estação conectada à rede física e responsável pela comunicação com todos as UMs dentro de uma determinada área de abrangência chamada *célula*;
- Célula: área geográfica sob a responsabilidade de uma MSS;
- Canal sem fio: canal de comunicação entre um MSS e uma UM ou ainda entre duas UMs. Este canal não utiliza meios guiados (cabos) e sim onda de rádio para a comunicação.
- *Hand-off*: processo de transferência de responsabilidade de uma MSS para outra durante o movimento de uma UM. Este processo engloba toda a troca de mensagens para a delegação de responsabilidades entre as MSS envolvidas.

Resumindo, um ambiente de computação móvel consiste de dois conjuntos distintos de entidades: as unidades móveis e as estações fixas. As unidades móveis (UMs) têm a capacidade de se movimentar irrestritamente dentro de uma célula ou entre duas células enquanto mantém a sua conexão de rede. Além disso, as UMs podem conectar-se a redes sem fio de diferentes locais a qualquer momento.

As estações fixas, como o próprio nome sugere, são aquelas conectadas a uma rede fixa, e algumas delas, chamadas de estações de suporte à mobilidade, MSS (*Mobile Support Station*), possuem uma placa de interface *wireless* para a comunicação com as unidades móveis.

Através das UMs, os usuários podem requisitar informações às MSSs e receber respostas em tempo real durante a sua movimentação, ou até mesmo executar estas transações localmente na unidade móvel. Através de consultas, usuários podem acessar dados através de conexões *wireless* ou na própria estação móvel. Estas consultas são denominadas *transações móveis*. A utilização do canal de comunicação *wireless* e a mobilidade dos usuários introduziram novas técnicas na área de sistemas de bancos de dados.

2.2 Arquitetura de Banco de Dados em Ambiente Wireless

As principais funcionalidades existentes nos bancos de dados para os computadores da rede fixa devem existir também nos bancos de dados *wireless*. Mesmo desconectado da rede, o banco de dados *wireless* deve permitir ao usuário consultar as informações contidas em *cache* ou mesmo realizar transações.

A Figura 1 apresenta um modelo geral para sistemas de banco de dados sem fio, similar ao descrito em [DUNHAM & KUMAR, 1999] para sistemas de computação móvel com infra-estrutura. Neste modelo, são considerados um *servidor de banco de dados* e um *banco de dados* para cada unidade fixa. Um servidor de banco de dados que suporta operações básicas de uma transação, como *read* (leitura), *write* (escrita), *prepare* (prepara), *commit* (consolida) e *abort*.

As dificuldades impostas pelo ambiente móvel induzem a variância da disponibilidade da rede e dos recursos computacionais. Estas restrições afetam diretamente as arquiteturas e os modelos de transações móveis. Os modelos de computação móvel devem garantir um ambiente eficiente para as aplicações existentes e para as novas aplicações. Em [PITOURA & SAMARAS, 1998] são propostos três modelos para computação móvel: o modelo *cliente/servidor*, o modelo *simétrico* e o modelo de *agentes móveis*.

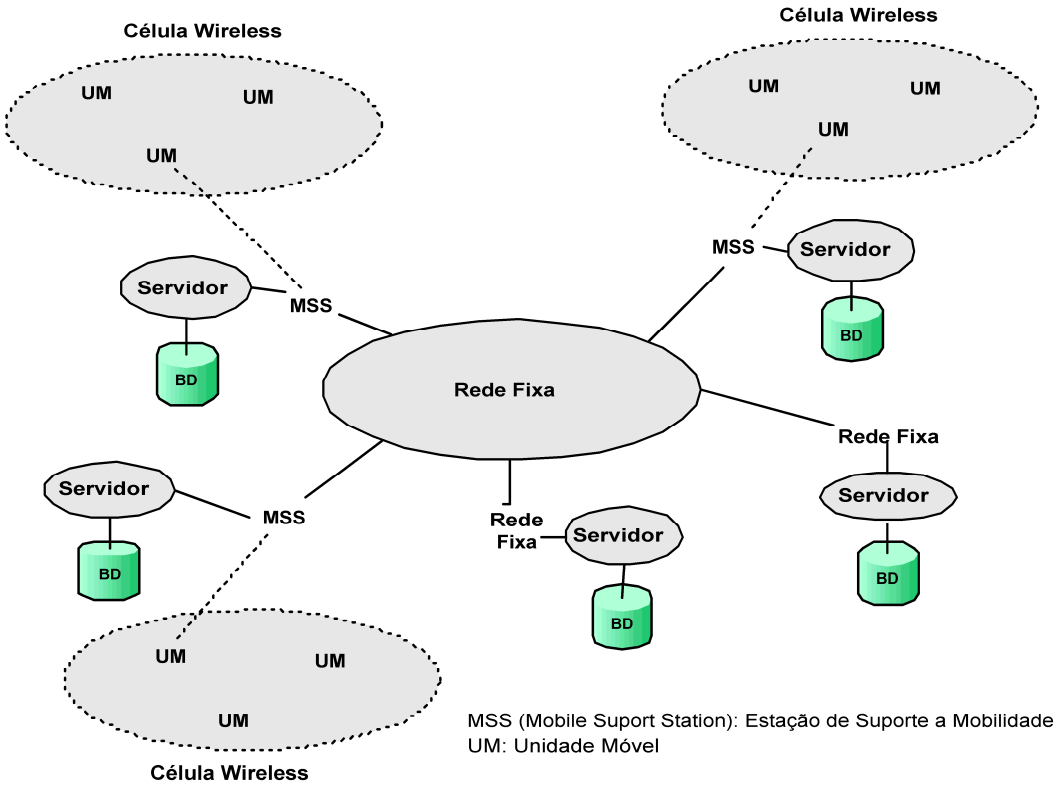

Figura 1 - Arquitetura Geral de Banco de Dados *Wireless*.

2.2.1 Modelo Cliente/Servidor

No modelo tradicional cliente/servidor, o servidor é responsável pelo gerenciamento dos dados enquanto o cliente pela interface com o usuário. As mensagens são enviadas para o servidor através de algum mecanismo de comunicação, por exemplo, RPC (Remote Procedure Call). As mensagens são interpretadas pelo servidor que, por sua vez, retorna o resultado do processamento para o cliente. O processamento pode ser desde uma simples pesquisa no banco de dados como uma transação mais complexa. Assim, todas as transações são realizadas no servidor, não dando autonomia para o cliente.

Para o ambiente de computação móvel, os SGBD-W (Sistema Gerenciador de Banco de Dados *Wireless*) podem ser divididos em: modelo *cliente/servidor estendido*, o modelo *cliente/agente/servidor* e o modelo *cliente/agente/agente/servidor*, como pode ser visto na Figura 2 nesta respectiva ordem.

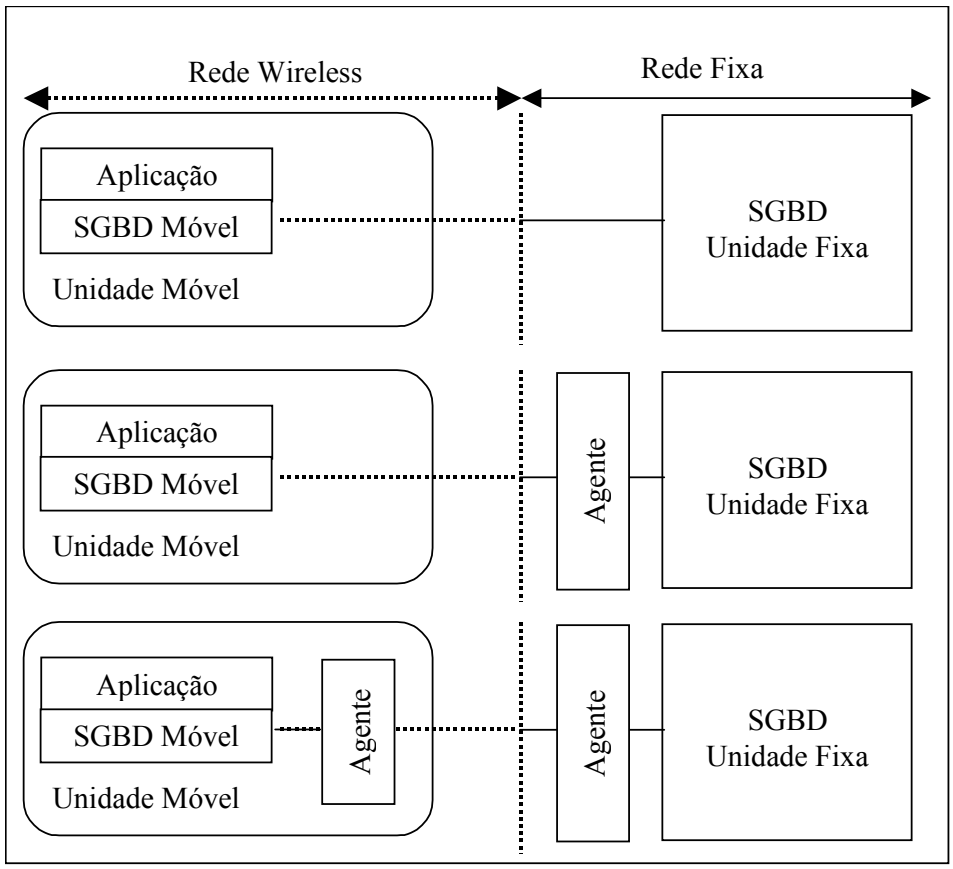

Figura 2 - Modelos Cliente/Servidor para SGBD-W, [PITOURA & SAMARAS, 1998].

2.2.1.1 Modelo Cliente/Servidor Estendido

O modelo cliente/servidor tradicional não atende as necessidades dos bancos de dados wireless. No modelo cliente/servidor tradicional tanto as máquinas clientes como as máquinas servidoras estão conectadas à rede fixa de computadores. Acessos e direitos são dados aos usuários em modo conectado, *on-line*.

Todos os controles do banco de dados são garantidos, para cada cliente, através de uma sessão estabelecida no SGBD servidor. Tendo a sessão finalizada, liberam-se todos os recursos alocados. O término voluntário da sessão causa a efetivação da transação através de um *commit*, caso contrário será executando um *rollback* na transação ativa.

Estando no modo *conectado*, o cliente móvel se comporta como se fosse um cliente qualquer da rede fixa, visto que todas as mensagens são encaminhadas para o servidor de banco de dados da rede fixa. Assim, não é necessária nenhuma modificação para o tradicional modelo cliente/servidor.

No ambiente móvel, a possibilidade do cliente móvel desconectar-se da rede a qualquer instante é muito alta. Aspectos como o gerenciamento de energia, interferência, falta do sinal de comunicação ou mesmo um *handoff* pode causar uma desconexão involuntária da unidade móvel. A mudança na estrutura cliente/servidor tradicional faz-se necessária para que o SGBD-W (Sistema Gerenciador de Banco de Dados *Wireless*) possa continuar em atividade mesmo em modo desconectado, como ilustrado na Figura 3. Técnicas como filas de mensagens RPC devem ser implementadas para garantir transparência entre os SGBD e o SGBD-W.

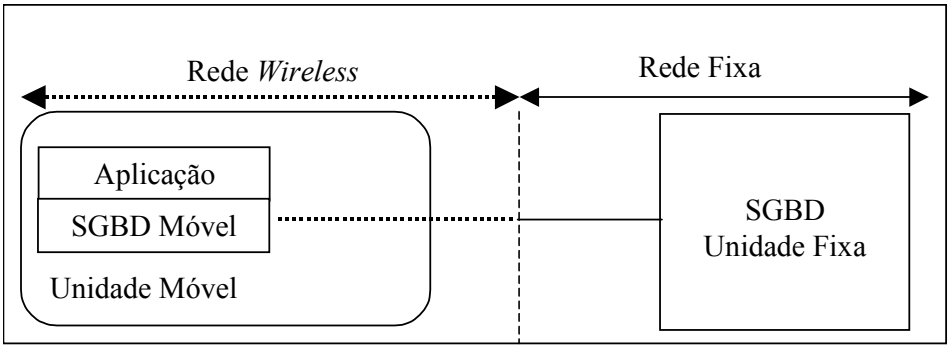

Figura 3 - Modelo Cliente/Servidor Estendido, [PITOURA & SAMARAS, 1998].

2.2.1.2 Modelo Cliente/Agente/Servidor

O modelo cliente/agente/servidor, conforme Figura 4, também pode ser encontrado na literatura como modelo em três camadas, *three-tier* ou *proxy*. Em redes com infra-estrutura os *proxies* podem estar localizados nas MSS (Mobile Station Support). Este modelo é mais apropriado onde existe um ambiente de vários bancos de dados, ou seja, um ambiente de banco de dados heterogêneos. O Agente pode fazer a tradução do protocolo utilizado entre o SGBD-W e o SGBD [PITOURA & SAMARAS, 1998].

Outra funcionalidade do agente está na otimização da largura de banda existente, efetuando a concentração e/ou compactação dos dados antes de serem transmitidos ao SGBD-W. Neste modelo o SGBD-W pode fazer uma solicitação ao agente e entrar no modo *doze* (adormecido) para economizar energia. O agente irá atender a solicitação do SGBD-W reunindo as informações necessárias. Somente após a execução da tarefa o *proxy* enviará o resultado para o SGBD-W. Técnicas específicas podem ser usadas de acordo com o tipo de trabalho a ser executado, tipo dos dados e do tipo da aplicação [PITOURA & SAMARAS, 1998].

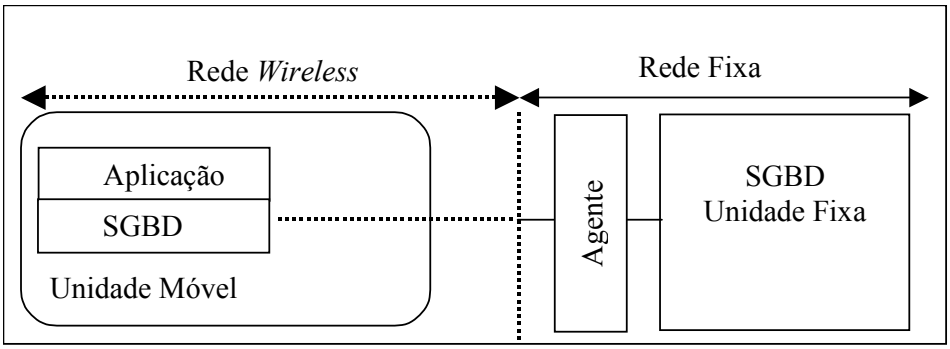

Figura 4 - Modelo Cliente/Agente/Servidor, [PITOURA & SAMARAS, 1998].

2.2.1.3 Modelo Cliente/Agente/Agente/Servidor

Neste modelo existe um agente localizado na unidade móvel e outro agente na rede fixa, conforme Figura 5. Eles estão entre os SGDB-W e o SGBD. O agente, localizado na unidade móvel, intercepta as chamados do cliente e juntamente com o agente servidor executam as otimizações necessárias para a redução dos dados no canal de comunicação. O agente cliente faz o papel de um servidor *proxy* para as aplicações na unidade móvel. Eles são totalmente transparentes para ambos os lados aplicação [PITOURA & SAMARAS, 1998].

A grande vantagem deste modelo está na abstração dos SGBD-W para com as técnicas de comunicação e redução dos efeitos do ambiente móvel. O modelo também apresenta escalabilidade de serviço e protocolos. Protocolos diferentes podem ser usados para assegurar a sincronização dos dados e de comunicação com o agente servidor, como HTTP (*HyperText Transfer Protocol*), FTP (*File Transfer Protocol*), entre outros.

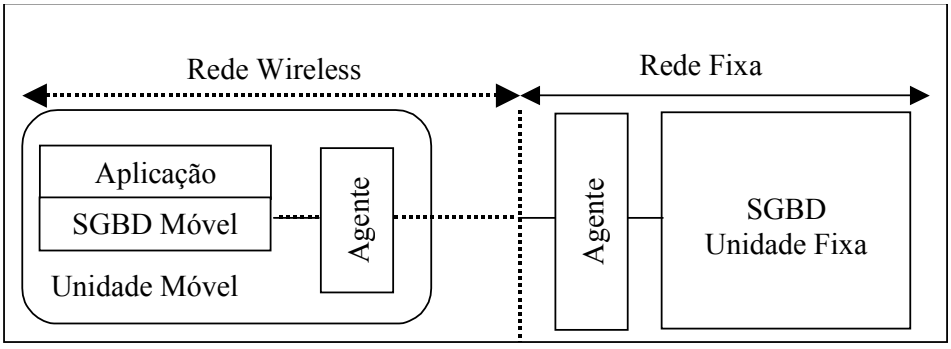

Figura 5 - Modelo Cliente/Agente-Agente/Servidor [PITOURA & SAMARAS, 1998].

2.2.2 Modelo Simétrico

No modelo simétrico os servidores fixos e os servidores móveis são considerados idênticos, conforme visto na Figura 6. Os SGBD-Ws possuem todas as funcionalidades de cliente e de servidor. Cada *site* tem ambos os serviços: de cliente e de servidor. Um SGBD-W pode comunicar-se com outros SGBD-Ws diretamente. Este é um caso de servidores de banco de dados totalmente distribuído. Clientes da rede fixa podem acessar as informações contidas no servidor móvel e vice-versa.

Nos modelos anteriores, o servidor localizava-se na parte fixa da rede. Assim os dispositivos móveis eram sempre clientes, operando como servidores somente no modo desconectado, *off-line*. Neste modelo o servidor pode estar localizado na estação fixa ou móvel, tanto para o modo conectado como no modo desconectado, sendo mais apropriado em ambientes onde as conexões entre as unidades móveis são mais estáveis, fortes, que as conexões com os servidores da rede fixa, caso das redes *ad-hoc*.

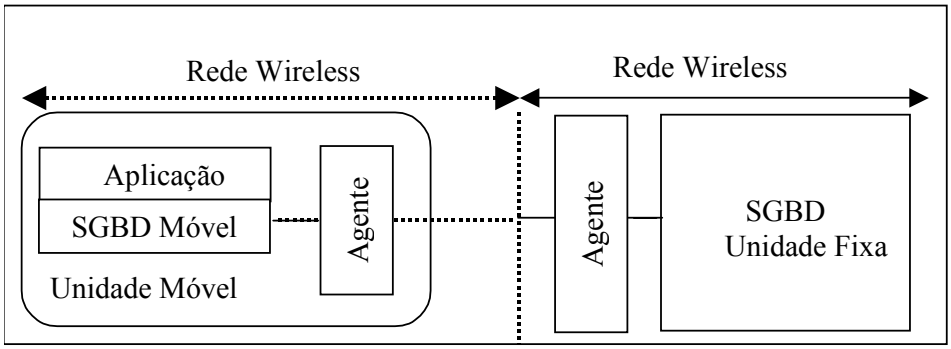

Figura 6 - Modelo simétrico [PITOURA & SAMARAS, 1998].

2.2.3 Modelo de Agentes Móveis

Os agentes móveis podem ser usados em conjunto com os modelos anteriores. Oferecem outras facilidades como acompanhar as unidades móveis, redefinindo suas atividades de acordo como o ambiente e ao longo do tempo. Este modelo tem como vantagem à economia de recursos da unidade móvel. Podem enviar agentes móveis para a rede fixa com determinada tarefa. Enquanto os agentes realizam seu trabalho a unidade móvel pode entrar no modo "*doze*" ou mesmo desconectar da rede. Assim, recursos da unidade móvel são economizados.

Segundo [CHESS et al., 1995], os agentes móveis são processos (ou objetos ativos) que tem a capacidade de migrar entre computadores de uma rede durante a sua execução, carregando consigo o seu estado de execução.

Percebe-se uma série vantagens com o uso dos agentes móveis, porém aspectos sobre segurança ainda merecem atenção, entre eles pode-se citar: a garantia que somente os agentes com origem autenticada possam executar ou a garantia que a execução do código agente não afete a máquina hospedeira.

2.3 Gerenciamento de Transações

Acompanhando o avanço das tecnologias de hardware para redes sem fio, surgiram sistemas baseados em computação móvel. Sistemas de bancos de dados também acompanharam essa tendência possibilitando a utilização de bancos de dados *wireless*. Essencialmente as operações realizadas em um banco de dados são chamadas de transações.

As propriedades de uma transação variam de acordo com seu ambiente (fixo, distribuído ou móvel). No ambiente de computação móvel, as limitações da rede e a mobilidade das unidades impõem a necessidade de mudanças na implementação eficiente do processamento de transações. Como as transações em ambientes móveis possuem um tempo de vida longo, essas transações são expostas a um grande número de

desconexões. Assim, as freqüentes desconexões são o maior obstáculo [DUNHAM et al., 1997], dificultando a manutenção das propriedades ACID.

Ao contrário de transações distribuídas, as transações móveis não são necessariamente originadas e terminadas no mesmo *site*. Um site é uma unidade, móvel ou não, que contém uma parte do banco de dados. Quando uma transação móvel migra de um computador para o outro, o estado da transação e seu progresso devem acompanhá-la.

A unidade móvel pode ter autonomia local para processar transações apesar das freqüentes desconexões. Na reconexão, os efeitos das transações móveis consolidadas durante a desconexão, devem ser incorporados ao banco de dados, garantindo desta forma, o sucesso da transação [WALBORN & CHRYSANTHIS, 1995].

Segundo [MADRIA, 1998] e [PITOURA & BHARGAVA, 1994], as transações móveis apresentam as seguintes características:

- As transações móveis devem dividir suas execuções em grupos de operações, sendo que alguns deles serão executados na unidade móvel enquanto outros nas estações fixas. E compartilha seus resultados parciais e finais com outras transações durante períodos de desconexões;
- Quando é usada uma conexão sem fio, as transações tendem a ser:
    - Monetariamente caras;
    - Possuir longa duração, devido a demoras de transmissão da própria rede;
    - Propensas a erros, devido as freqüentes desconexões e também porque as unidades móveis são mais vulneráveis a acidentes;
    - Baseadas em sessão. Pelo alto custo, por exemplo, de conexão celular, os dados são enviados em sessões.

A mobilidade resulta em transações que:

- Acessam sistemas heterogêneos;
- Acessam dados locais (possivelmente imprecisos);
- As transações podem envolver dados que são dinamicamente realocados;

2.3.1 Propriedades das Transações

Na rede fixa, os usuários realizam transações consistentes e duráveis. Nestes ambientes as propriedades ACIDs são garantidas. A maioria das propostas abrangendo os modelos de transações, apresentadas na literatura, considera as transações móveis como sendo parte de uma transação onde existe alguma flexibilização nas propriedades ACID.

As transações possuem quatro importantes propriedades que devem ser satisfeitas [DATE, 2000]:

- Atomicidade: As transações são atômicas, isto é, ou completa a execução de todas as operações ou nenhuma alteração é realizada no banco de dados;
- Consistência: As transações preservam a consistência do banco de dados. Antes de iniciar a transação o banco de dados deve estar em um estado consistente e permanecendo assim após a execução da transação;
- Isolamento: As transações são isoladas uma das outras. Existem várias transações ocorrendo simultaneamente no banco de dados, porém os dados que elas estão atualizando devem estar isolados um do outro, isto é, duas transações distintas não podem estar atualizando o mesmo item de dados em transações diferentes;
- Durabilidade: Depois de efetivada as transações, elas devem permanecer no banco de dados mesmo ocorrendo uma falha no sistema;

2.3.2 Tipos de Transações

Vários modelos de transação têm sido propostos na literatura para o suporte às características e problemas da computação móvel. Cabe a cada modelo o gerenciamento das transações móveis e a escolha do tipo de transação a ser utilizado. O objetivo comum desses modelos é o provimento das propriedades ACID. Os tipos de transações influenciam o modo de execução de cada modelo, constituindo um tópico de extrema importância no projeto de banco de dados sem fio.

Transações podem ser classificadas sob vários critérios. Um destes critérios é a **duração das transações**, ou seu **tempo de vida**:

- **Transações de Curta Duração - TCD** são caracterizadas pelo tempo curto de execução (na ordem de poucos segundos) e por terem acessado a uma parte relativamente pequena do banco de dados.
- **Transações de Longa Duração – TLD**, por outro lado, são caracterizadas por um longo período de tempo de execução (sendo mensurado em minutos, horas ou até mesmo dias) e acessam a uma grande porção do banco de dados.

Transações também podem ser classificadas de acordo com sua **estrutura**. São diferenciadas e divididas em três categorias: transações simples (*flat*), transações *aninhadas fechadas* e transações *aninhadas abertas.*

2.3.2.1 Transações Simples

Esta é a transação típica suportada pela maioria dos SGBDs, e consiste, simplesmente, em delimitar o conjunto de operações pertencentes a uma transação. O nível de aplicação é que determina onde começa e onde termina uma transação. A designação de transação simples (*flat)* vem, precisamente, do fato de só existir um nível de controle, ou seja, o sucesso da transação depende da execução de todas as operações. A Figura 7 mostra a estrutura de uma transação simples , sendo T a transação, e X,Y e Z operações.

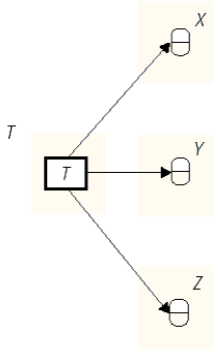

Figura 7 - Arquitetura de uma Transação Simples

2.3.2.2 Transações Aninhadas

No modelo de transações aninhadas, as transações podem ser vistas como uma árvore de transações planas, conforme ilustra a Figura 8. Uma transação raiz, chamada de transação pai, é dividida em várias outras subtransações, chamadas de transações filhas. Por sua vez as subtransações podem ser divididas novamente em outras subtransações e assim por diante. Apenas a transação do nível folha contém operações sobre os dados, enquanto as transações dos nós intermediários fornecem uma espécie de controle para seus descendentes. Pode-se dizer que as transações aninhadas constituem um conceito mais generalizado de transação.

O nível de aninhamento é geralmente *aberto*, permitindo que as próprias subtransações possam também ter suas subtransações. Nesta taxonomia, a diferença entre transações aninhadas abertas e fechadas é dada pelas características de suas terminações. Em ***transações aninhadas fechadas*** o funcionamento é hierárquico, onde a subtransação (filha) começa sua execução depois da subtransação pai ter começado e termina sua execução antes de seu pai. A subtransação filha só é consolidada se o pai também for. Os resultados parciais não são visíveis fora da transação. Já nas **transações aninhadas *abertas*** *os* resultados parciais de suas subtransações são visíveis para outras transações aninhadas. De acordo com [PITOURA & BHARGAVA, 1994] o modelo de *transação aninhada aberta* é a mais apropriado para modelar as transações móveis. Transações Particionadas (Split Transactions) são um exemplo de transações aninhadas abertas.

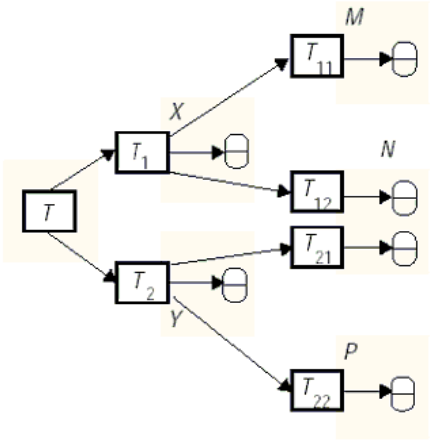

Figura 8 - Estrutura de uma Transação Aninhada

2.4 Controle de Concorrência

O controle de concorrência é a parte de um SGBD com os objetivos de assegurar que transações executadas em nós múltiplos produzam os mesmos resultados como se fossem executadas seqüencialmente num nó simples e manter o isolamento das transações e a consistência dos dados. O controle de concorrência possibilita o compartilhamento de dados de forma transparente aos usuários.

Quando diversas transações são executadas de modo concorrente em um banco de dados, a propriedade do isolamento pode não ser preservada. Sendo necessário um controle para a interação de transações concorrentes. Em ambientes móveis, o poder de processamento das unidades móveis é reduzido, contribuindo para que o tempo de execução de uma transação seja mais longo. Outro fator importante é que algumas unidades ficam temporariamente indisponíveis, ou seja, em estado desconectado. Neste caso a transação não pode ser considerada como falha e ser abortada.

A seguir são apresentados alguns conceitos básicos importantes no contexto de controle de concorrência em bancos de dados móveis.

- **Integridade de Dados:** Deve-se assegurar que as informações que entram no sistema são verdadeiras e estão representadas corretamente. Um sistema bem desenvolvido deve permitir que se imponham restrições (regras para manutenção da integridade dos dados).
- **Consistência de Dados**. A mesma informação é guardada em diversos registros, possivelmente em arquivos diferentes. No entanto, seu valor deve ser o mesmo a qualquer momento em todos esses registros. Pode-se ter uma situação de inconsistência quando vários usuários estão utilizando concorrentemente o banco de dados, enquanto um usuário está alterando alguns registros, um ou mais usuários estão tendo acesso aos mesmos registros, sendo que alguns ainda não foram alterados.
- **Objetos**. Menor unidade do banco de dados acessível pelo sistema de controle de concorrência. Os objetos podem ser arquivos, páginas, registros, etc.

- **Ação**. Comando de processamento primitivo e indivisível, o qual é executado por um usuário simples.
- **Operação**. Seqüência de ações executadas sobre um objeto.
- **Transação**. É usada para descrever uma seqüência de operações sobre um ou mais objetos do banco de dados, transformando o atual estado consistente do sistema num novo estado consistente.

- **Escalonador de Transações.** Controla o acesso de transações aos objetos do banco de dados. O escalonador é responsável pela ordem na qual as diferentes operações para um conjunto de transações são feitas.

- **Anomalias no Processamento Concorrente de Transações.** Resultado da interferência entre os usuários que estão simultaneamente tendo acesso ao banco de dados. Essa interferência causa problemas de inconsistência, caracterizada pela violação das restrições de integridade.

2.4.1 Teoria da Serialização

Presume-se que cada transação preserva a consistência se é executada sozinha. Assim uma seqüência serial de transações também preserva consistência. Sabe-se que uma execução sucessiva de transações corresponde a um tipo de escalonamento especial que não apresenta anomalias, e que exclui as transações concorrentes. O problema de controle de concorrência consiste em só permitir a execução dos escalonamentos que não gerem anomalias.

A execução de transações concorrentes deve ser realizada em modo isolado, não permitindo que os dados utilizados por uma transação estejam disponíveis a outras transações enquanto sua execução não for completada. Considera-se que a execução concorrente de uma transação deve deixar o banco de dados em um estado que seja equivalente ao de sua execução seqüencial [ÖZSU 99].

A teoria da seriabilidade é utilizada como um critério pra determinar a correção de execuções concorrentes de múltiplas transações em um sistema gerenciador de banco de

dados. Para transformar um escalonamento qualquer num escalonamento serializado, é preciso conhecer as características dos escalonamentos em série.

Um escalonamento de execução, denominado *S*, é definido em [ÖZSU & VALDURIEZ, 1999] sobre um conjunto de transações T = {T_1, T_2 …,T_n}. Sobre este conjunto será especificada uma ordem de intercalação de execução dessas transações.

São definidos dois tipos de transação, de acordo com [ÖZSU & VALDURIEZ, 1999] Transações de Leitura e Transações de Escrita. Transações de Leitura são aquelas que realizam operações de leitura em um item de dados *x* – *Read*(x). Transações de Escrita realizam operações de escrita em um item de dados *x* – *Write*(x). Por definição, se duas operações quaisquer $O_{ij}(x)$ e $O_{kl}(x)$ acessam o mesmo item de dados *x*, elas são consideradas *conflitantes* se ao menos uma destas operações for uma operação de escrita Write(x) ou W(x). É importante salientar dois pontos nesta definição: transações de leitura, não causam conflitos e duas operações podem pertencer a mesma transação ou a transações diferentes.

Neste último caso, diz-se que estas transações estão em *conflito*. Intuitivamente, a existência de um conflito entre duas operações indica que suas ordens de execução são importantes. A ordem de execução de operações de leitura é insignificante [ÖSZU 99].

Pode-se salientar dois tipos de conflitos, quanto à concorrência. Conflito *leitura-escrita* (ou *escrita-leitura*) e conflito *escrita-escrita*. Caso as operações pertençam a duas transações distintas então serão chamadas de *transações conflitantes*.

A definição formal de um escalonamento completo, S_T^c é definida sobre um conjunto de transações T = {T_1, T_2 …,T_n} como uma ordem parcial $S_T^c = \{\sum T, \prec T\}$ onde

1. $\Sigma_T = \bigcup_{i=1}^{n} \sum_i$.
2. $\prec T \supseteq \bigcup_{i=1}^{n} \prec i$

3. Para quaisquer duas operações O_{ij}, $O_{kl} \in \Sigma_T$,tanto $O_{ij} \prec T \; O_{kl,}$ ou $O_{kl} \prec T \; O_{ij}$

A primeira condição explicita que o domínio do escalonamento é a união dos domínios das transações. A segunda condição define a relação de ordem como um super conjunto de relações de ordem de transações individuais. Isto mantém a ordenação das operações em cada transação. A condição final simplesmente define a ordem de execução entre as operações conflitantes.

Um escalonamento seria, por exemplo:

$T_1 = R_1(x), W_1(x),$

$T_2 = R_2(x), W_2(x),$

$T_3 = R_3(x), W_3(x),$

O escalonamento S é serial quando $S_1 = R_1(x), W_1(x), R_2(x), W_2(x), R_3(x), W_3(x)$, isto é, $\mathbf{T_1 \rightarrow T_2 \rightarrow T_3.}$

Um escalonamento concorrente é considerado correto, se ele é computacionalmente equivalente a um escalonamento serial. Mecanismos de controle de concorrência são corretos se as transações executam as operações em um escalonamento serial.

Propriedades de um escalonador:

- Uma transação somente deverá ler um item de dados se este item já tiver sido escrito e consolidado por uma transação (*commit*);
- Nenhum item de dados deve ser lido ou escrito enquanto a transação ainda não estiver consolidada;
- Precisa garantir a consistência

3 Modelos de Transações Móveis

As desconexões das estações móveis por longos períodos de tempo e limitações da rede, sugerem reconsiderações dos modelos de transação e técnicas de processamento de transações. Assim, muitas propostas de modelos de transações têm surgido, todas elas com diferentes noções de transação móvel [SEYDIM, 1999]. Muitas destas propostas visualizam uma transação móvel como um conjunto de subtransações. O gerenciamento destas transações pode ser estático, na própria estação móvel ou no servidor de banco de dados, ou pode se mover de uma estação base para outra, da mesma forma que uma unidade móvel [DUNHAM et al., 1997].

Unidades móveis podem submeter transações em dois modos:

1- uma transação inteira pode ser submetida em uma única mensagem, a unidade móvel entrega o controle de execução a seu coordenador e espera o retorno dos resultados da execução da transação.
2 - as operações de uma transação são submetidas em múltiplas mensagens de requisições.

Enquanto a primeira abordagem envolve um único coordenador para todas as operações de uma transação, a segunda pode envolver múltiplos coordenadores, devido à mobilidade das unidades. Por exemplo, uma unidade móvel pode passar a uma célula nova, e depois disto obter os resultados de operações previamente submetidas. Na célula atual, submeterá o restante das operações de transação ao *coordenador* na MSS respectiva.

Uma unidade móvel pode mover-se a qualquer hora, inclusive para longe de sua célula atual, depois de submeter uma operação e antes de receber uma resposta do coordenador. Neste caso, procedimentos adicionais são necessários para localizar a unidade móvel e obter os resultados de operações submetidas a ela. Por simplicidade, assume-se que uma unidade móvel só pode mover-se de sua célula atual depois que

receber os resultados para todas as operações submetidas naquela célula. Porém, acredita-se que o método discutido poderá ser aplicável com a incorporação de alguns procedimentos [JING 94].

Desconexões de rede não podem ser tratadas como falhas, por isso, se métodos precisarem completar uma tarefa, os dados devem estar presentes localmente, no dispositivo móvel. Devido às tradicionais técnicas de serialização (ex., monitoramento de transações e escalonadores) não funcionarem adequadamente em um ambiente desconectado, novos mecanismos precisaram ser desenvolvidos para o gerenciamento do processamento de transações móveis.

Nas seções seguintes são abordados alguns modelos de transações móveis. São eles [SEYDIM 1999] [ALVARADO 2001]:

- Modelo de Transações Canguru *(Kangaroo Transaction)*;
- Modelo de Transações baseado em *Clustering*;
- Modelo de Transações *Pro-Motion*;
- Modelo de Transações *Reporting;*
- Modelo de Transações MDSTPM;
- Modelo de Transações baseada em Semântica;
- Modelo de Transações usando Replicação em Two-Tier (duas fileiras);

3.1 Modelo de Transações Canguru

Definido por [DUNHAM et al., 1997] este modelo propõe um modelo de transação que focaliza o comportamento do movimento da unidade móvel durante a execução de transações. Transações móveis são denominadas neste modelo, como *Transações Canguru,* pois incorporam a propriedade de "saltar", em um ambiente móvel, de uma estação base para a outra (*handoff*) conforme o movimento da unidade móvel.

A estrutura básica é composta principalmente de Transações Locais (TL) para um particular SGBD. Usa-se TL para identificar a transação tradicional. Para alguns

sistemas gerenciadores de banco de dados, executa-se TL como uma transação local. As operações normalmente executadas são: leitura (*read*), escrita (*write*), inicio de transação (*begin transaction*), cancelamento de transação (*abort transaction*) e finalização de transação (*commit transaction*).

A transação global TG é composta de subtransações que podem ser vistas como transações locais TL de um banco de dados, sendo cada uma destas subtransações TL uma seqüência de operações OP.

Transações móveis são geradas na unidade móvel e são completamente executadas em um sistema de banco de dados distribuído na rede fixa, conforme ilustra a Figura 9. A TC propõe a implementação de um *Agente de Acesso a Dados* (AAD) no topo de um Sistema Global de Banco de Dados (SGBD) existente. Este agente estará em todas es estações base e irá gerenciar as transações móveis e o movimento da unidade móvel. Quando este agente receber uma requisição de transação de um usuário móvel, o AAD transfere esta requisição para a estação base ou para a unidade fixa que possuir o dado requerido. O AAD age como um *gerenciador de transações móveis* e *como um coordenador de acesso* a dados para o *site*.

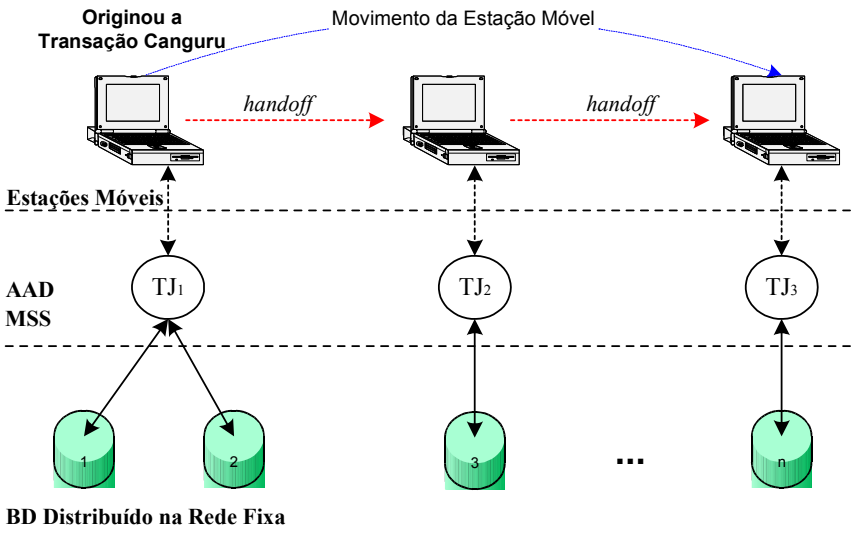

Figura 9 - Exemplo de TM no modelo Canguru.

Cada subtransação representa uma unidade de execução em uma estação base e é chamada de Transação *Joey* (TJ), ou Transação Filha como é também conhecida [DUNHAM et al., 1997]. As seqüências de transações globais e locais que são executadas abaixo de uma dada TC são chamadas de "bolsa", seguindo a mesma analogia. Primeiramente uma estação base cria uma TJ para sua execução. Uma TC possui um único número de identificação consistindo do número da estação base e uma única seqüência de números identificando a transação. Quando a estação móvel migra de uma célula para outra, o controle da TC muda para um novo AAD, de uma outra estação base. O AAD da nova estação base cria uma nova TJ como resultado do processo de *handoff*.

O Gerente de Transação Móvel (GTM) mantém uma tabela de estado de transação na estação base e o AAD mantém o estado dessas transações. Também é mantido um arquivo de *log* local no qual o GTM escreve os registros que são precisos para propósitos de recuperação, mas o *log* não contém qualquer registro relacionado à recuperação das operações de banco de dados. A maioria dos registros armazenados no arquivo de *log* é relacionada ao estado da TC e algumas informações para compensação da transação, caso seja necessário. Durante o processo de *handoff* o arquivo de *log* precisa ser analisado para assegurar recuperação se uma falha ocorrer durante este processo.

Alguns registros que são armazenados no arquivo de *log* são:

- Quando a transação Canguru é inicializada um registro de BTTC (Begin Transaction Transação Canguru) é escrito no arquivo de *log;*
- Durante o processamento de um *handoff* (antes), um registro de HOTC (HandOff TC) é escrito na MSS que originou o *handoff*;
- Enquanto um registro de CTTC (ConTinua TC) é escrito na MSS de destino.
- As TJ's são registradas como *Begin* (BTTJ) e *End* (ETTJ).
- Subtransações com TJ tem um registro de (BTST) e (ETST), *Begin* e *End-Transaction* respectivamente.

Registro	Conteúdo
BTTC	IDTC, Modo
CTTC	IDTC, Modo
BTTJ	IDTJ, AntIDTJ
BTST	IDST, Requisição, Compensável
ETTJ	IDTJ, ProxIDTJ
ETST	IDST
ETTC	IDTC
HOTC	IDTC

Tabela 1 - Registros contidos no arquivo log.

O modelo de transação Móvel Canguru foi construído baseado nos modelos de Transações Aninhadas Abertas [1]e Transações Particionadas ou *Split Transactions*[2]. E os modos de processamento deste modelo de transação são o modo de *Compensação* e o modo *Particionado*.

Quando uma transação canguru é executada no *modo de compensação*, o cancelamento (*abort*) de uma Transação *Joey* fará com que seja desfeita esta subtransação e todas as outras subtransações TJ da transação canguru. Transações *Joey* já consolidadas terão que ser compensadas.

No *modo particionado* se uma subtransação falhar, outras já consolidadas, não serão compensadas. Transações filhas ativas podem terminar ou ser desfeitas, a decisão é a cargo do SGBD local. Quando uma requisição de transação é feita por uma unidade móvel o AAD da estação base associada cria uma transação canguru para realizar esta requisição. Uma identificação de Transação Canguru (ID_TC) é criada para identificar a transação. É criada uma TJ via operação de particionamento (*split*). Este processo é

[1] Cada uma das subtransações de uma transação passa a ser uma transação por si só, podendo ser finalizada ou desfeita autonomamente das outras transações. O nível de aninhamento *aberto* permite que as próprias subtransações possam também ter suas subtransações e assim por diante.

[2] Transações Particionadas (*Split Transactions*) são um exemplo de transações aninhadas abertas.

dinâmico. Uma nova TJ é criada somente quando a estação móvel se movimenta para uma nova célula, ocasionando um *handoff*. Como parte deste procedimento uma operação de particionamento é sempre executada.

Conforme a Figura 10, a TC é iniciada na unidade móvel:

- Uma TJ é iniciada na MSS, estação base, conectada;
- As transações executam na rede fixa como transações aninhadas abertas.
- Se a unidade móvel muda de localização, a JT é particionada e uma nova JT entra em execução na nova MSS.
- A JT_1 pode efetuar uma operação de *commit* independentemente da TJ_2.

Se a TJ_1 falhar:

- Modo de Compensação: é desfeita toda a TC.
- Modo Particionado:
  - TJ's previamente consolidadas não são compensadas;
  - Nenhuma nova JT é inicializada;
  - Transações *filhas* são consolidadas ou abortadas dependendo da decisão do SGBD local.

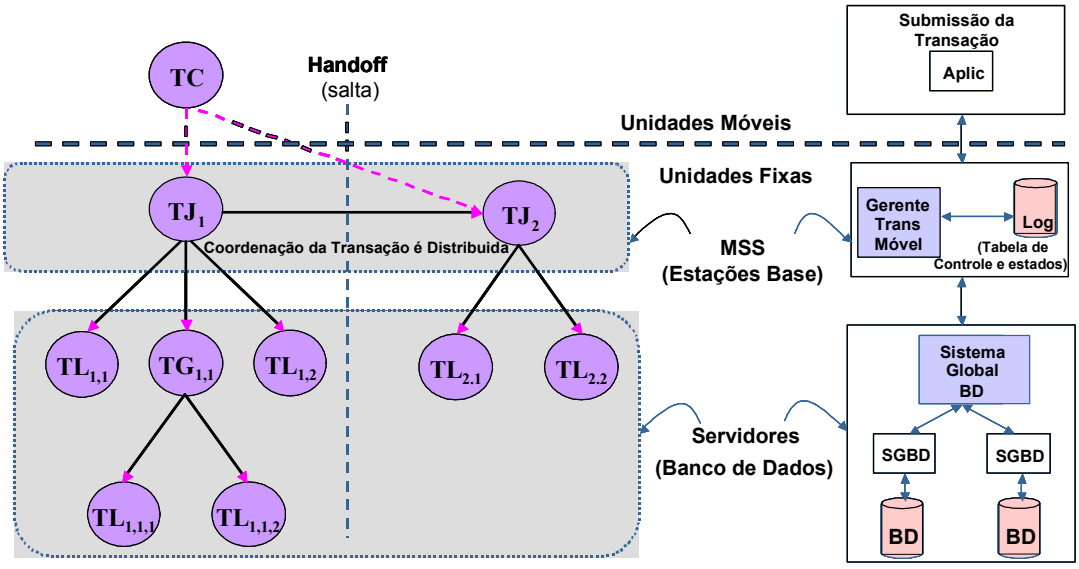

Figura 10 - Estrutura Geral de uma possível TC [ALVARADO et al., 2001].

Como mostra a Figura 10 uma divisão (particionamento) é executada primeiro, durante o salto da MSS 1 para a MSS 2, a TC é então dividida em duas subtransações: TJ1 e TJ2. É assumido que as subtransações da transação móvel (isso é as transações globais e locais) são executadas em seqüência. A próxima subtransação não é requisitada até que a primeira seja previamente completada. Assim, assume-se que as transações locais e globais abaixo de TJ1 acontecem antes das que estão abaixo de TJ2. Isto garante que TJ1 preceda TJ2 em ordem serializável. Assume-se que nenhuma subtransação será criada debaixo de TJ2 até o que a TJ1 que está sendo executada seja consolidada.

Transações Canguru utilizam o mesmo mecanismo de controle de concorrência que as transações de *multibanco de dados*, no caso o bloqueio de cada subtransação que está em execução. Podendo ser utilizado o mecanismo *de controle de concorrência de bloqueio em duas fases.*

O fluxo de controle de processamento de uma TC (Transação Canguru) pelo GTM (Gerente de Transações Móveis) é descrito a seguir:

1. A unidade móvel requisita uma Transação Canguru, o correspondente AAD (Agente de Acesso a Dados) da estação base, passa a transação para o seu GTM, para gerar um identificador único (ID_TC) e cria uma entrada na tabela de estados. O GTM também cria a primeira TJ para executar localmente em sua célula. No final desta etapa, um registro de BTTC é escrito no arquivo de *log* do GTM.
2. Com a criação da TJ pelo GTM, é criado um ID_TJ e uma entrada na tabela de estados. O contador de número de TJ's na tabela de estados é incrementado de 1. Um registro de BTTJ é escrito no *log*. Uma entrada de TJ é escrita na tabela de estados das transações.
3. A Transação Canguru é executada no contexto das Transações Joey. Para cada TJ, uma tradução é feita para mapear as operações da TC em específicas transações locais e globais. Baseado nos resultados da tradução, uma entrada de ST (Subtransação) é criada na tabela da transação para cada transação nativa (local ou global). Um registro de BTST é escrito no *log* do GTM.

4. Quando um *handoff* ocorrer, o AAD é imediatamente avisado, e inicia o protocolo de *handoff* na camada de transação. O AAD inicia a execução da operação de *split* (particionamento) no *site* de origem. Como parte deste particionamento o AAD escreve um registro de HOTC no arquivo de *log*, para indicar que um *handoff* ocorreu. O *log* do *buffer* é passado para o disco. Estas ações representam uma operação de *checkpoint*.
5. A estação base (MSS) de destino então inicia a outra parte da operação de particionamento. Um registro de CTTC é escrito e depois uma nova TJ é criada com o conteúdo sendo o resto da TC. Uma entrada TC é iniciada na tabela de estado do destino.
6. Para assegurar a atomicidade do sistema fonte SGBD local e global, eles irão determinar quando as operações de *commit* ou *abort* serão executadas na subtransação. Quando o AAD é notificado (pelo SGBD) que uma subtransação foi consolidada, o AAD escreve um registro de ETST no arquivo de *log*. Além disso, se uma TC está em modo de processamento particionado, então uma entrada de ST é removida da tabela de estados. Se estiver em modo de processamento de compensação, a entrada continua, pois a transação poderá ser abortada e uma transação de compensação deverá ser executada. Se não existir nenhum registro de ST para esta TJ, então um registro de ETTJ é escrito no *log* e o estado da tabela da tabela de estados da JT é mudado para o *commit*. Finalmente, o contador para o número de transações Joeys ativas é decrementado de 1 unidade. O estado da tabela TC para a última MSS ativa contém os valores correntes para o contador Joey. Se o contador Joey for igual a zero e o estado da TC for de *commit*, então a TC é consolidada.
7. Quando o usuário móvel indicar que a transação chegou ao fim, a entrada na tabela de estados é modificada para *commit*. Neste instante, se o contador da transação *Joey* ativa for igual a zero a TC é consolidada.
8. Para consolidar uma TC, a entrada ETTC é escrita no *log* e todas as tabelas de estados para todas as MSS envolvidas são liberadas.

Registro	Atributos	Descrição
TC	ID_TC	Identificador da TC
	Modo	Particionado (Split) ou Compensação
	Contador Joey	Contador das TJ's ativas desta TC
	Estado	Ativo, *Commit* ou *Abort*
	PrimeiraIDTJ	Ponteiro para o primeiro registro de TJ para esta TC
TJ	IDTJ	Identificador da TJ
	ProxIDTJ	Ponteiro para o próximo registro de TJ para a TC
	AntIDTJ	Ponteiro para o registro anterior de TJ para a TC
	Estado	Ativo, *Commit* ou *Abort*
	STList	Lista de subtransações locais ou globais
	Compensável	Sim ou não
ST	IDST	Identificador da ST
	Estado	Ativo, *Commit* ou *Abort*
	Requisição	Requisição de TG ou TL
	Compensável	Sim ou não
	CompTR	Transação Compensável

Tabela 2 - Entradas da Tabela de Estados de uma TC [ALVARADO et al., 2001].

3.2 Modelo *Clustering*

Um flexível modelo de consistência de "dois níveis" foi introduzido em [PITOURA & BHARGAVA, 1995] [PITOURA & BHARGAVA, 1999] para tratar as freqüentes desconexões (previsíveis e variáveis) das unidades móveis. É um modelo de consistência de dois níveis, pois permite a execução de operações tanto em um banco de dados consistente, bem como, concede também, em virtude das limitações da computação móvel, um certo limite de inconsistência.

Este modelo consiste basicamente em particionar um banco de dados em *clusters*, ou seja, em agrupar semanticamente os dados relacionados ou aproximadamente localizados, com o objetivo de formar os *clusters*. O agrupamento, *clustering,* pode ser baseado na localização física dos dados. Dados localizados muito próximos ou fortemente conectados tendem a pertencer ao mesmo *cluster*, enquanto que dados em unidades remotas, ou desconectadas, são pertencentes a *clusters* separados. A configuração dos *clusters* é feita de forma dinâmica. Para tirar vantagem da natureza previsível das desconexões, *clusters* podem ser criados seguidos de uma desconexão ou conexão da unidade móvel associada. Além disso, acomoda as unidades móveis que migram de um cluster para outro quando entram em uma nova célula.

Por outro lado, os agrupamentos de dados, *clustering*, podem ser definidos usando a semântica de dados, como os dados locais, ou definindo um perfil do usuário. Dados de localização, que representam o endereço de uma unidade móvel, são dados que rapidamente variam e são replicados para muitos locais. Conseqüentemente estes dados são, freqüentemente, imprecisos, e quando suas cópias são atualizadas criam um *overhead* muito alto. Isto pode não ter necessidade para prover consistência para este tipo de dados. Por outro lado, definindo perfis do usuário para a criação de um *cluster*, pode ser possível diferenciar os usuários, baseado nas exigências de seus dados e aplicações. Por exemplo, pode ser considerado que dados que são acessados freqüentemente por algum usuário ou dados que são privados a um usuário, pertencem ao mesmo agrupamento, independente do seu local ou semântica.

Alguns dados são armazenados em uma unidade móvel, para que possa suportar uma autonomia durante as desconexões, ele próprio se torna um *cluster*. Para cada banco de dados duas cópias são permitidas, umas delas, chamada de *versão rígida*, deve ser globalmente consistente, e a outra, denominada de *versão fraca*, pode tolerar alguns níveis de inconsistência, mas deve ser localmente consistente.

Os itens de dados são relacionados por restrições, chamadas de **restrições de integridade**, que expressam o relacionamento dos dados que o banco deve satisfazer. Restrições de integridade entre itens de dados pertencentes ao mesmo *cluster* são chamadas restrições *intra-cluster*, e devem ser completamente consistentes. E entre *clusters* diferentes são denominadas restrições *inter-cluster*, que permitem níveis de inconsistência por estarem em clusters diferentes. O grau de inconsistência pode variar dependendo da disponibilidade da largura de banda da rede entre os clusters.

Durante desconexões ou quando a conexão estiver fraca, os dados que o usuário móvel poderá acessar serão aqueles que satisfazem as restrições *intra-cluster*. Para maximizar o processamento local e reduzir o acesso a rede o usuário interage com um *cluster* disponível localmente.

As transações neste modelo podem ser *rígidas* ou *fracas*, [PITOURA & BHARGAVA, 1995]. **Transações fracas** podem acessar somente versões fracas, ou seja, versões localmente consistentes. Enquanto que as **transações rígidas** acessam versões rígidas, versões globalmente consistentes. A execução das transações rígidas acontece quando as unidades móveis estiverem fortemente conectadas, enquanto que nas transações fracas podem acontecer quando as unidades móveis estão desconectadas do banco de dados.

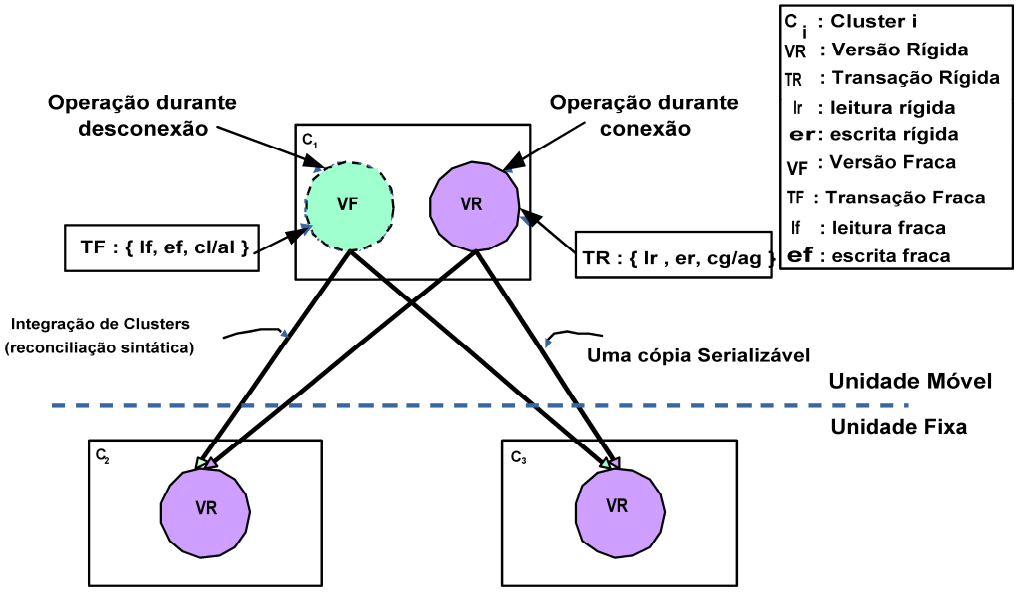

Figura 11 - Exemplo de TM no modelo Clustering [ALVARADO et al., 2001].

As operações de leitura e escrita também são classificadas como *fracas e rígidas*. Às operações fracas são permitidas a acessar somente elementos de dados pertencentes ao mesmo *cluster,* porque suportam o modo desconectado. E às operações rígidas é permitido o acesso a o banco de dados global. Para cada item de dados duas cópias são mantidas, uma delas rígida (cópia global) e a outra fraca (cópia local).

Este modelo utiliza o **protocolo de commit em duas fases**, ou seja, faz *commit* local no modo desconectado e o *commit* global é obtido através da reconciliação sintática, [GRAHAM et al., 1995], das transações fracas, como visto na Figura 11.

Para cada transação (T) é definida uma ordem parcial de operações (OP, <), onde OP é o conjunto de leituras e escritas, fracas ou rígidas, *aborts*, e *commit* local executados pela transação, e **<** representa a sua ordem de execução.

A ordem parcial deve especificar a ordem das operações de dados conflitantes e contém uma operação de *abort* ou *commit* que estão por último nesta ordem. Duas operações fracas (rígidas) conflitam se elas acessam a mesma cópia de item de dados e se no mínimo uma destas operações for uma operação de escrita fraca ou rígida. As transações fracas operam somente com OP fracas, enquanto que transações rígidas operam somente com OP rígidas.

Para implementar a semântica mencionada é necessário que cada gerenciador de transação local no cluster C_i mantenha duas versões, x^f_i e x^r_i , para cada cópia x_i. A versão x^r_i é atualizada por transações rígidas e é chamada de **versão rígida**. Enquanto que x^f_i é atualizada tanto por transações rígidas como fracas e é denominada **versão fraca**. Transações rígidas lêem x^r_i, enquanto que transações fracas lêem x^f_i.

3.3 Modelo *Pro-Motion*

É um modelo de processamento de transação móvel que suporta o processamento das transações na unidade móvel (local) em modo desconectado. Foi proposto por [WALBORN & CHRYSANTHIS, 1999] e [MAZUMDAR & CHRYSANTHIS, 1999], teve como o objetivo à migração de aplicações de banco de dados existentes e suporte ao desenvolvimento de novas aplicações envolvendo acesso a dados móveis.

Neste modelo foi introduzido o uso de *compacts*. *Compacts* são as unidades básicas de armazenamento e controle que permitem que a execução de transações móveis seja feita localmente, na unidade móvel. As informações necessárias para gerenciar os *compacts* são encapsuladas nele. Para melhorar a autonomia e aumentar a concorrência, objetos semânticos são utilizados na construção dos *compacts*, sempre que for possível.

O modelo de execução do *Pro-Motion* usa o modelo de transação do tipo aninhada particionada (*Split-Nested*). Este modelo considera o sistema móvel como se fosse uma transação de longa duração (TLD), executada no servidor. Os recursos necessários para a criação dos *compacts* são obtidos por estas transações através de operações usuais do banco de dados (leituras e escritas).

Um outro aspecto importante do modelo PRO-MOTION, é a introdução de um subsistema de gerenciamento de transações, consistindo de:

- Um *Gerenciador Compact* no servidor de banco de dados;
- Um *Agente Compact* na unidade móvel;
- E um *Gerenciador de Mobilidade* executando na estação base.

A construção dos *compacts* é de responsabilidade do **gerenciador de *compacts*** no servidor de banco de dados. O gerenciador de *compacts* irá funcionar como um *front-end* para o servidor do banco de dados e aparenta ser um cliente de banco de dados executando uma única Transação de Longa Duração.

Em cada unidade móvel, o *agente compact* (AC) é responsável pelo gerenciamento de *cache*, também pelo processamento da transação, controle de concorrência, arquivo de *log* e recuperação. O *gerenciador de mobilidade* é responsável pela troca de transmissões entre agentes, conforme mostra a Figura 12. É importante salientar que neste modelo as transações na unidade móvel são executadas localmente mesmo em modo conectado.

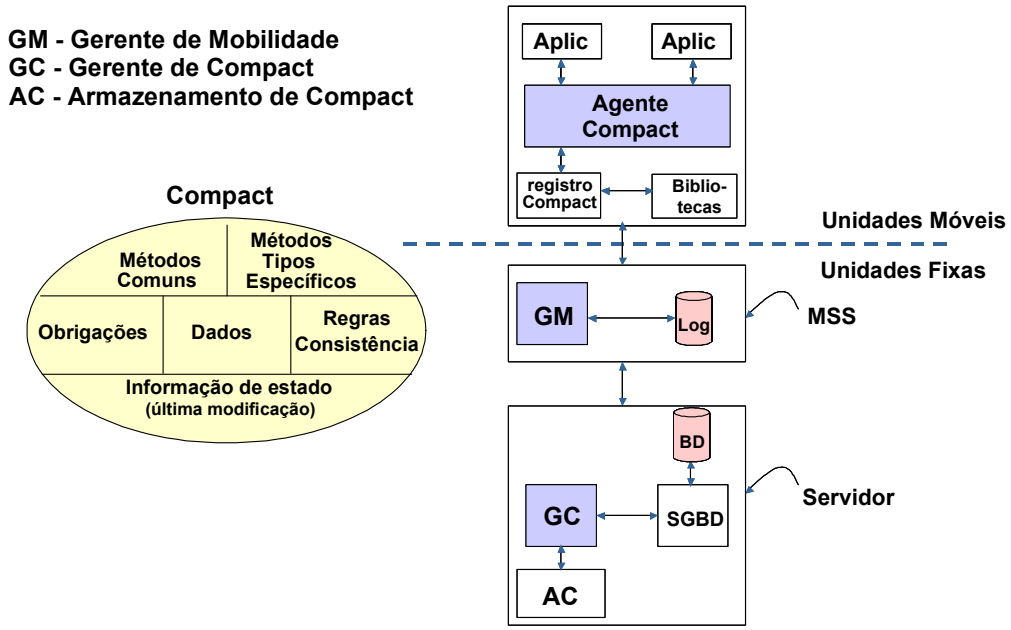

Figura 12 -Estrutura do modelo Pro-motion [ALVARADO et al., 2001].

Um processo de sincronização é executado pelo AC e o gerenciador de *compact* na reconexão. Este processo verifica os *compacts* modificados por transações localmente finalizadas (*commited*). Se os *compacts* preservam consistência global, então um *commit* global é executado. A arquitetura do Sistema *Pro-Motion* é mostrada na Figura 13.

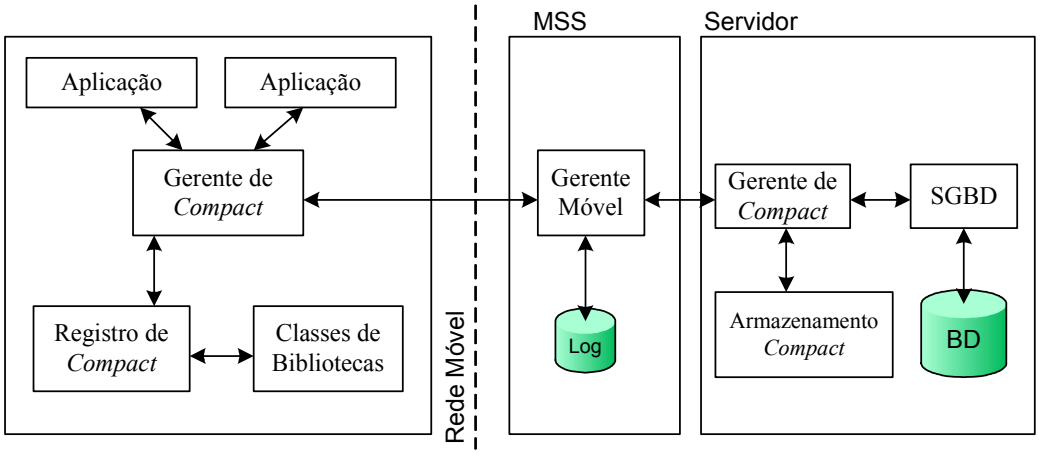

Figura 12 - Arquitetura do Pro-Motion [WALBORN & CHRYSANTHIS, 1999].

Segundo [MAZUMDAR & CHRYSANTHIS, 1999], no primeiro estágio de ressincronização, o registro do *compact* acha os IDs (identificadores) de todos os

compacts que foram modificados quando a unidade móvel estava desconectada. Se todos os *compacts* são válidos, a ressincronização pode ser completada e podem passar ao estágio final. Porém, caso algum *compact* esteja expirado, um trabalho adicional deverá ser efetuado antes da ressincronização.

Segundo estágio, o registro de *compact* transmite os IDs dos *compacts* modificados expirados, tenta renovar cada *compact* e estende seu *deadline* (prazo final).

Caso nenhuma outra entidade tenha bloqueado ou modificado os dados do *compact*, o GC, gerenciador de *compact,* pode restabelecer um *compact* e validar do lado da unidade móvel, como se o *compact* nunca tivesse expirado. Se todos os *compacts* expirarem serão restabelecidos, e a atualização pode ser feita na sua totalidade e a ressincronização procede para o estágio final.

Porém, se os *compacts* modificados não podem ser restabelecidos, os valores originais válidos ou restabelecidos terão que ser obtidos do gerente de *compact*. Os *compacts* que não poderem ser restabelecidos serão inválidados. Neste momento um evento de *log* pode ser refeito para todas as transações consolidadas. Se uma transação lê ou modifica um *compact* inválido, a transação é considerada abortada e todos os *compacts* modificados pela transação abortada são marcados como indisponíveis. Assim como, as transações que lerem transações indisponíveis serão abortadas e os *compacts* marcados como indisponíveis.

Quando um evento de *log* for completamente refeito, os *compacts* passam a ser válidos e podem ser incorporados ao servidor de banco de dados.

No último estágio da ressincronização o registro de *compact* inspeciona o conjunto de *compacts* para os *compacts* que são válidos e modificados. Para cada *compact* consultado é gerado um evento OP que é retornado ao *compact* do lado do servidor para trazer os dados do servidor em acordo com os dados da unidade móvel. Quando todas as operações forem comunicadas, a unidade móvel manda mensagem de *commit* para o gerente de *compact*.

No servidor, o conjunto de operações de atualização no processo de ressincronização é dividido em uma única transação, que está consolidada no servidor. Além disso, o gerenciador de *compacts* tenta manter os bloqueios em todos os itens já utilizados. Se os bloqueios são retidos pelo gerente de *compact*, no lado do servidor, os *compacts* associados e os *compacts* no lado da unidade móvel permanecem válidos e utilizáveis.

Se os bloqueios não puderem ser mantidos, o *compact* no lado do servidor e os *compacts* no lado da unidade móvel são invalidados. Uma vez que a ressincronização é completada, os *compacts* são liberados e a unidade móvel volta ao estado de *hoarding*.

Os *compacts* são representados neste sistema como objetos, que encapsulam [MAZUMDAR & CHRYSANTHIS, 1999]:

- Os dados armazenados;
- Métodos (códigos) para o acesso aos dados armazenados;
- Informações sobre o estado atual do *compact*;
- Regras de consistência, se existirem, que precisam ser seguidas para garantir a consistência global dos itens de dados;
- Obrigações, tais como um prazo limite que cria intervalos de tempo para cada recurso que é mantido pela unidade móvel;
- Métodos que fornecem uma interface na qual a unidade móvel pode gerenciar o *compact*.

O gerenciamento dos *compacts* é um esforço cooperativo do servidor do banco de dados e da unidade móvel. Os *compacts* são obtidos a partir do banco de dados via requisições do computador móvel quando uma demanda real ou antecipada é criada. Os *compacts* são periodicamente atualizados como resultado do processamento na unidade móvel.

Sempre que a unidade móvel necessita de itens de dados, esta manda uma requisição deste item de dado para o servidor de banco de dados. Se este dado estiver disponível para satisfazer a requisição, o servidor do banco de dados cria um *compact* que é

transmitido a uma unidade móvel para fornecer os dados e métodos necessários para a execução local da transação. Uma vez recebido o *compact* pela unidade móvel, ele é gravado em um *registro de compacts* (RC) que é usado pelo *AC* para obter a informação de localização e o estado de todos os *compacts* abertos.

Cada objeto, *compact,* possui uma interface comum que é usada pelo *AC* para gerenciar os *compacts* listados no *RC.* O Conjunto de métodos básicos necessários para gerenciar os *compacts* incluem:

- **inquire**, é o método no qual, o AC pode obter o estado corrente do *compact*;
- **notify**, permite ao *compact* receber uma notificação quando o estado da UM é modificado;
- **dispatch**, usado para operações de processo no *compact*;
- **commit**, para fazer as operações, de uma transação especificada, permanente no banco de dados; e
- **abort**, para abandonar as modificações feitas nos dados do *compact* por uma dada transação.

3.4 Modelo de *Reporting*

Neste modelo, a origem das transações é representada em termos de relatórios, *reporting* e *co-transações* [CHRYSANTHIS 1991, 1993], as quais podem ser executadas tanto nas estações bases como nas unidades móveis. Uma transação *reporting* pode compartilhar seu resultado parcial com a transação de origem a qualquer hora e pode realizar a operação *commit* independentemente.

Uma co-transação é uma classe especial da transação anterior, na qual pode ser forçada a esperar por outra transação. Depois de entregar seu resultado, pode continuar a executar outras operações.

Segundo [CHRYSANTHIS 1991, 1993], este modelo analisa transações aninhadas e transações aberto-aninhadas, mostrando as limitações de seus ambientes móveis. Considerando um ambiente de banco de dados sem fio (*wireless*) como um sistema de multibancos de dados especial, com exigências específicas de transações na unidade móvel, que são consideradas como um conjunto de subtransações. Neste modelo é proposta a transação aberto-aninhada que suporta atomicidade, transações não-compensáveis e dois tipos adicionais, *reporting* e *co-transações* .

Transações aninhadas fechadas consolidam em ordem inversa, de baixo para cima até a raiz, ou seja, da última subtransação até a primeira. Conseqüentemente, uma subtransação aninhada começa depois da transação que a originou e termina a sua execução antes. E a consolidação da subtransação é condicionada à consolidação de seu pai. O aninhamento aberto é mais flexível quanto à restrição de atomicidade do nível superior da transação aninhada fechada, onde os resultados só podem ser vistos após a consolidação da transação pai. A transação aninhada aberta permite que seus resultados parciais sejam observados fora da transação. Enquanto em execução, as transações podem compartilhar os resultados parciais e parcialmente podem manter o estado de uma subtransação móvel (executada na unidade móvel) na sua estação base MSS.

Em transações do tipo *aninhado* se estende o conceito de atomicidade, permitindo o cancelamento de uma subtransação sem que, toda a transação seja abortada. No sistema de transações aninhadas, as transações possuem uma estrutura interna. Esta estrutura pode ser descrita como hierarquia de subtransações dispostas em formato de árvore. Todas as transações, menos uma, que é a primordial, são subtransações de uma outra. A estrutura hierárquica determina o tipo de relacionamento entre transações: uma subtransação que se encontra imediatamente abaixo de outra é chamada de filha desta; duas subtransações filhas de uma mesma transação mãe são chamadas descendentes na hierarquia que liga a primeira à segunda. Arquitetura do modelo Reporting é ilustrada na Figura 14.

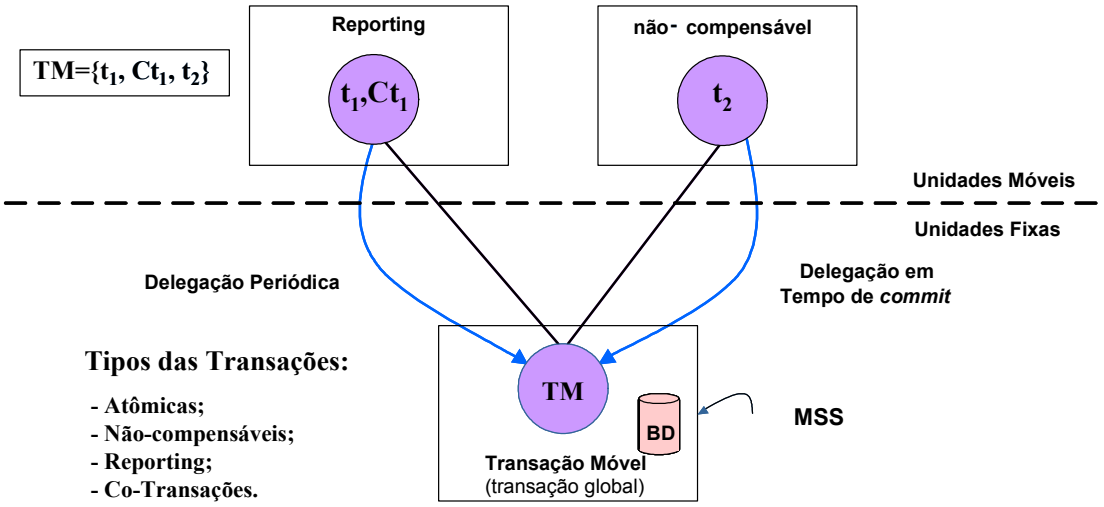

Figura 13 - Exemplo de TM no modelo Reporting [ALVARADO et al., 2001].

Uma transação móvel é estruturada como um conjunto de transações, algumas das quais são executadas na unidade móvel. Assim como nos outros modelos de transações móveis, as limitações das unidades móveis requerem que parte do estado da computação assim como armazenamentos sejam feitos nas estações base.

Neste modelo são propostos quatro tipos de transações aberto aninhadas, são eles:

- *Transações atômicas* estão associadas com os eventos {Begin, Commit, Abort} tendo como propriedades padrão abort e commit.
- *Transações não-compensáveis* são componentes de transações que não são associados com uma transação de compensação[3] (transação de compensação atua de modo inverso em relação à transação a qual ela está associada). Transações não-compensáveis podem ser consolidadas a qualquer hora, desde que elas não possam ser compensadas, não lhes é permitido consolidar os efeitos

3 O trabalho de uma transação de compensação é reverter o efeito de uma transação consolidada de uma forma que não dependa do que aconteceu ao banco de dados entre o momento em que a ação foi executada e o momento em que a transação de compensação é executada [SILBERSCHATZ 99]. Transações compensatórias são utilizadas para desfazer transações confirmadas ou transações ativas que afetam outras transações, sem ser preciso utilizar abortos em cascata.

em objetos quando eles consolidam. Transações não-compensáveis são estruturadas como sub-transações (como em transações aninhadas).

- *Transações reporting* - informam a outra transação alguns dos seus resultados a qualquer ponto durante a execução. Um relatório pode ser considerado como uma delegação de estado entre transações. Assim, as transações reporting são associadas com a primitiva da transação de Relatório, além das operações primitivas Begin, Commit e Abort.
- *Co-transações* se comportam como co-rotinas nas quais o controle é passado de uma transação para outra na hora de compartilhar os resultados parciais. Co-transações estão suspensas na hora de delegação e eles retomam a sua execução quando eles recebem um relatório. Assim, transações não-compensáveis são opostas as co-transações, pois estas não podem ser executadas concorrentemente.

3.5 Modelo MDSTPM

Gerenciamento do Processamento das Transações de Banco de Dados Múltiplos (*Multidatabase Transaction Processing Manage Arquitecture*), [YEO & ZASLAVSKY, 1994] propõe um *framework* para suportar submissões de transações de uma unidade móvel em um ambiente de *banco de dados múltiplos*. Um sistema de múltiplos bancos de dados consiste de um número de integrantes autônomos de sistemas de gerenciamento de banco de dados e cada um desses SGBDs é responsável pelo gerenciamento das transações localmente.

Para facilitar a execução de transações globais (TG), uma camada de *software* adicional pode ser implementada, permitindo o escalonamento e coordenação de transações através desses componentes de sistema de banco de dados múltiplos, ou seja, a existência de um MDSTPM em cada unidade móvel.

Este modelo torna possível o gerenciamento de estações móveis e as transações globais submetidas. Uma vez que a TG foi enviada, o *site* coordenador pode então escalonar e coordenar a execução desta TG de acordo com as necessidades da estação móvel.

As principais motivações para o desenvolvimento desta estratégia, segundo [YEO & ZASLAVSKY, 1994], foram:

- O usuário da estação/unidade móvel pode se desconectar da rede por tempo indeterminado e desenvolver suas tarefas, sem ter que esperar a execução completa de uma TG;

- Os outros computadores são conectados com uma rede de comunicação confiável e desta maneira são menos suscetíveis à falhas de rede.

O modelo MDSTPM consiste dos seguintes componentes como mostra a Figura 15:

- O *Gerenciador de Comunicação Global (GCG*) é responsável pela geração e gerenciamento das filas de mensagens dentro do site local. Além disso, ele faz a comunicação, a entrega e troca de mensagens entre as unidades participantes da rede;

- O *Coordenador Gerente de Transações Globais (CGTG*). O coordenador da transação global é o site em que a transação global iniciou-se. E é o coordenador das transações globais;

- *Participantes Gerenciadores de Transações Globais (PGTG*). São todos os gerenciadores participantes da transação global;

- O *Gerenciador de Transações Globais (GTG)*. O CTG coordena o envio de subtransações globais para os seus sites coordenadores. O CTG pode ser categorizado como Subgerente Global de Concorrência (SGC) e Subgerente Global de Escalonamento "Scheduling" (SGE);

- *Subgerente Global de Concorrência (SGC*). O SCG é responsável pela aquisição dos requerimentos de controle de concorrência necessários para o sucesso da execução de subtransações e transações globais;

- *Subgerente Global de Escalonamento "Scheduling"* (SGE). O SGE é responsável pelo escalonamento das transações globais de das subtransações globais;
- O *Gerenciador de Transações Locais (GTL*) tem a responsabilidade pela a execução e recuperação das transações executadas localmente;

- O *Gerenciador de Recuperação Global (GRG*) coordena a consolidação e a recuperação de subtransações e das transações globais depois de uma falha. Ele assegura que os efeitos das subtransações globais, depois de finalizadas sejam escritos no banco de dados local ou se algum "abort" ocorrer, que todo o processo seja desfeito;

- *Gerenciador de Interface Global (GIG)* coordena o envio de requisições e respostas entre o MDSTPM e o gerente do banco de dados local que pode estar executando em um sistema de banco de dados relacional ou orientado a objetos.

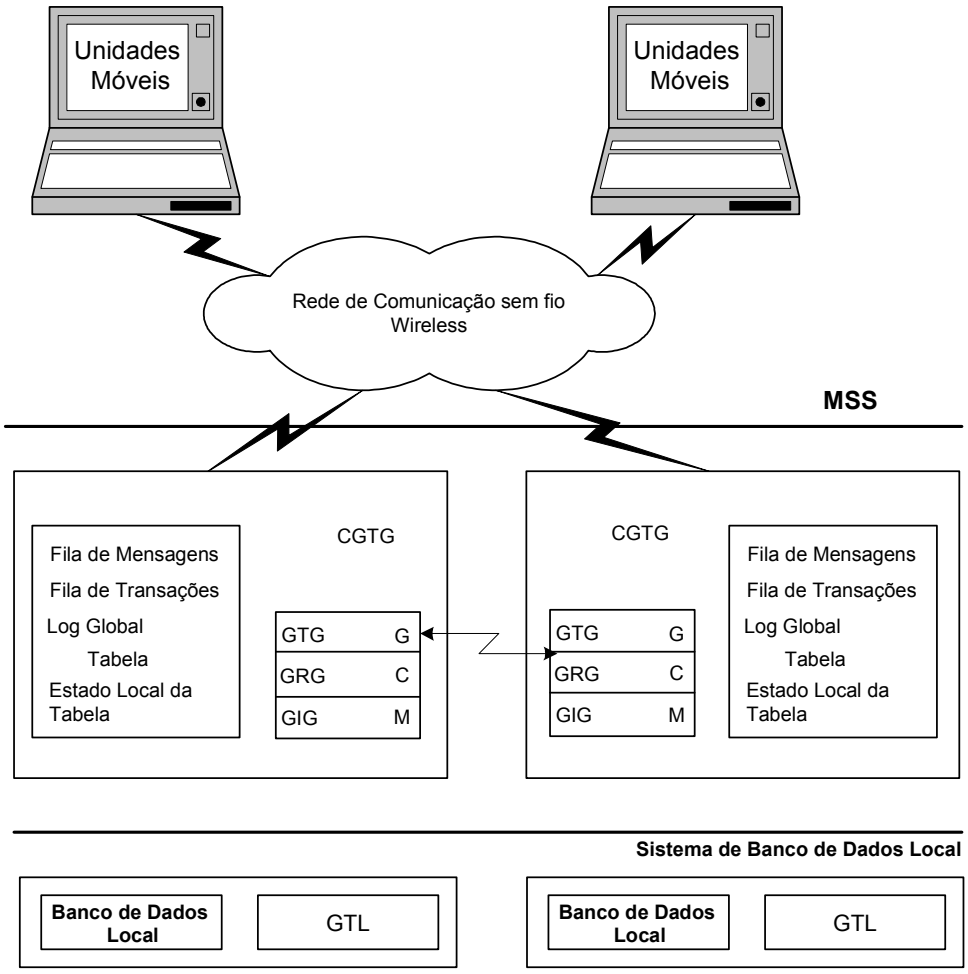

Figura 14 - Arquitetura do modelo MDSTPM [YEO & ZASLAVSKY, 1994].

Assume-se que cada cliente móvel submete uma transação a um agente coordenador como mostra a Figura 15. Uma vez que a transação foi submetida, o agente coordenador programa e coordena sua execução de acordo com o interesse do cliente móvel. Unidades móveis podem desconectar voluntariamente da rede antes de ter qualquer transação associada completa. A arquitetura deve satisfazer o seguinte:

- Prover o gerenciamento de *framework* de transação de forma que os usuários e programas aplicativos poderão acessar dados por transparência de múltiplos locais;

- Aumentar a concorrência de banco de dados e disponibilidade de dados pela adoção de um controle de concorrência distribuído e mecanismo de recuperação que preserve a autonomia local;
- Implementar o conceito de extensibilidade para suportar vários sistemas de banco de dados dentro do *framework* de forma que os componentes possam cooperar com banco de dados orientado a objeto ou relacional;
- Prover um ambiente onde a transação proposta processe o componente operando independentemente e transparente ao SGBD local;
- Incorporar o conceito de computação móvel pelo uso de estações móveis de trabalho.

O MDSTPM não pode ser classificado como um SGBDM. Trata-se de um framework para gerenciamento de transações distribuídas com alguma extensão para as transações móveis. Assim, este modelo pode ser utilizado para qualquer aplicação usuária de uma arquitetura totalmente distribuída, ou seja, tem-se um SGBD autônomo em todas as unidades participantes, fixas ou móveis.

Em um ambiente de múltiplos bancos de dados, *multidatabase systems* (MDS), é mais complexo assegurar a serialização da execução concorrente das transações globais devido ao fato que o GTL (Gerente de Transações Globais) possui autonomia e pode implementar também diferentes mecanismos de controle de concorrência. Portanto um dos maiores problemas encontrados para manter a serialização das transações globais é como assegurar a ordem de execução de transações globais nos sistemas de banco de dados locais, SGBD.

A abordagem utiliza o gerenciamento de unidades móveis e transações globais submetidas a estas unidades móveis para fazer parte do MDS durante suas conexões com um respectivo coordenador de seu nodo. Uma vez que uma transação global foi submetida, o coordenador local pode escalonar e coordenar a execução da transação global em nome da unidade móvel. Deste modo, a unidade móvel pode desconectar da rede sem esperar uma transação global para completar.

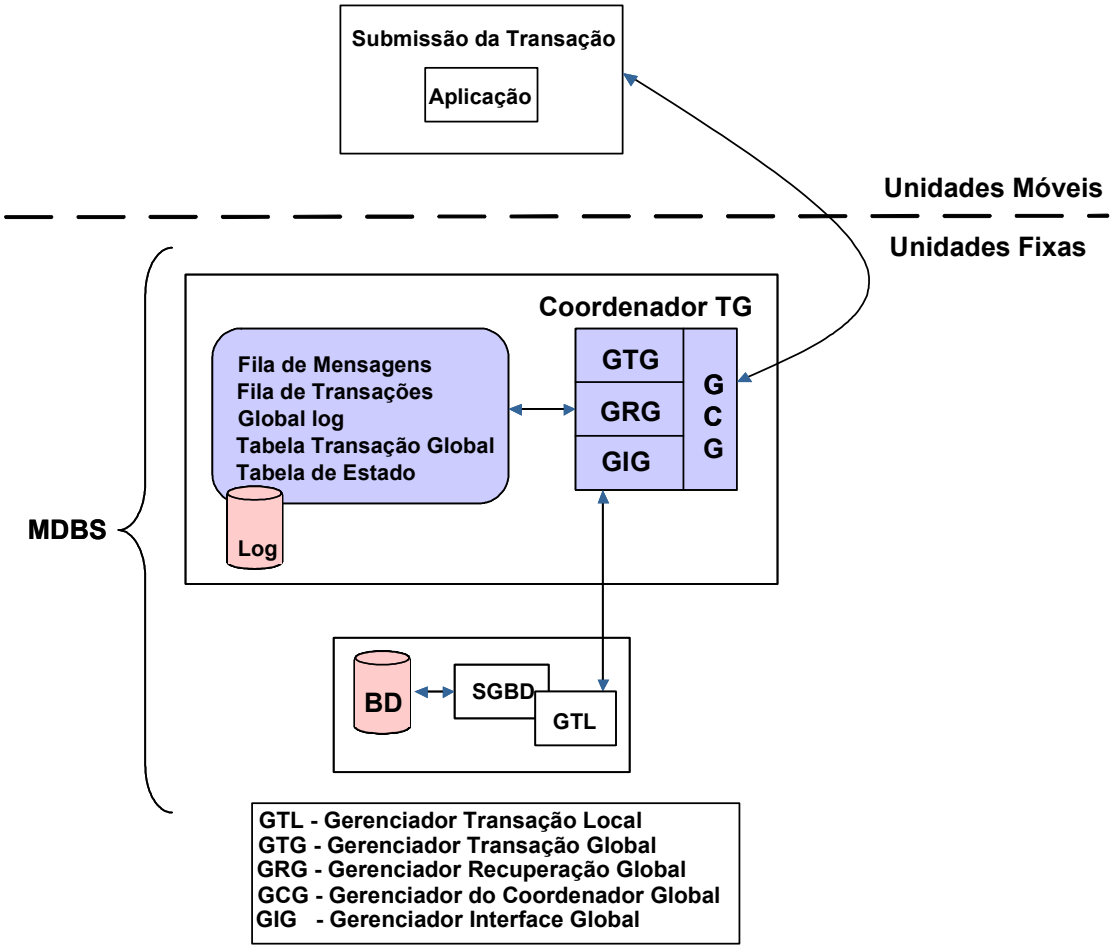

Figura 15 - Arquitetura do Sistema MDSTPM [YEO & ZASLAVSKY, 1994].

Uma abordagem alternativa denominada *Message and Queuing Facility* (MQF) é proposta para facilitar a implementação da estratégia escolhida pelo MDSTPM para a entrega e recebimento de mensagens. Uma estação móvel, denominada por [YEO & ZASLAVSKY, 1994] como *mobile workstation* (*mws*), envia uma mensagem de requisição, juntamente com a informação requerida para processamento, para o seu nó coordenador já associado. As mensagens são então manuseadas de forma assíncrona permitindo que a estação móvel se desconecte da rede para desenvolver outras tarefas, deixando que o nó coordenador execute as transações globais submetidas conforme seu interesse.

Como estas estações móveis possuem um período de conexão intermitente e curto, esta estratégia de MQF é apropriada para este ambiente devido aos seguintes fatores:

- Pela simplicidade do gerenciamento de entrega e recebimento de mensagens;
- Pela independência de tempo de desconexão da unidade móvel, ela pode estar desconectada da rede por tempo indeterminado enquanto as transações globais submetidas por estas estações estão sendo coordenadas e executadas pelo nó coordenador;
- Pela habilidade de cada estação consultar o estado de suas transações globais de acordo com sua conveniência.

Para gerenciar as transações submetidas por unidades móveis, é proposta uma simples *transação global* utilizando mecanismo de *fila*. O princípio básico sob o mecanismo de fila está o conceito de máquinas de estados finitos. Porque pode ser definido um conjunto de possíveis estados e transições de um estado para outro durante o período de vida de uma transação global. Isto é, do período de vida das ações ou subtransações dentro das primitivas:

- BEGIN_GLOBAL_TRANSACTION (comece_transação_global) ; e
- END_GLOBAL_TRANSACTION (finalize_transação_global)

A chave para implementar o modelo proposto é projetar uma fila fácil de fazer que trace cada um destes estados. Existem cinco subfilas de transação que são usadas para administrar transações/subtransações globais submetidas ao sistema gerenciador de banco de dados local pela unidade móvel.

- **Fila de Entrada**. Esta fila contém todas as transações/subtransações globais que chegaram primeiro ao coordenador. Está em ordem crescente baseado no tempo de chegada da transação.
- **Fila de Alocação**. Serão selecionadas as transações/subtransações globais para execução baseado em *first-in first-out* (FIFO – primeiro-que-entra / primeiro-que-sai), ou uma prioridade baseada no algoritmo de escalonamento. Todos os bloqueios pedidos para transações/subtransações globais serão adquiridos durante esta fase.

- **Fila Ativa**. Esta fila contém transações/subtransações globais ativas atualmente.
- **Fila de Suspensão**. Esta fila contém todas as transações/subtransações globais que completaram a primeira fase do protocolo *two phase committ* (Protocolo de *commit* em duas fases [YEO & ZASLAVSKY, 1994].
- **Fila de Saída**. Esta fila contém todas as transações globais completadas.

Quando uma transação global está sendo submetida por uma unidade móvel, o GCG colocará a transação global na Fila de Entrada. Periodicamente, o GSS efetua uma varredura na Fila de Entrada e seleciona uma transação global para execução. Uma vez selecionada, a transação global será transferida para a Fila de Alocação onde todos os pedidos de bloqueio serão adquiridos por GCS. Então a transação global será transferida à Fila Ativa onde subtransações globais são executadas em outros locais e serão despachadas pelo GCG. Uma vez que a transação global completou a primeira fase do protocolo *two phase commit*, é colocada então na Fila Suspensa e, para concluir o *two phase commit* a transação global é colocada então na Fila de Saída.

Uma vez que uma transação foi selecionada para execução pelo GSS, é necessário um controle de concorrência que é adquirido e gerenciado pelo GCS. Para assegurar a consistência dos objetos de banco de dados na execução entrelaçada de múltiplas transações concorrentes, um mecanismo de controle de concorrência é usado para isolar os efeitos de uma transação sobre as outras transações executadas simultaneamente. Segundo [FERREIRA & FINGER, 2000], tal protocolo é baseado na existência de tabelas de bloqueio, que faz com que haja um certo grau de replicação das informações já mantidas nos gerenciadores locais. Além disso, uma transação local só poderá emitir uma operação após obter o bloqueio de sua transação global imediatamente superior. A serializabilidade é o critério usado geralmente para designar o mecanismo de controle de concorrência.

De acordo com [YEO & ZASLAVSKY, 1994], um dos principais problemas encontrado para manter a seriabilidade das transações globais é como assegurar que a ordem de execução de transações globais é preservada pelo gerente de transação local (GTL). Alguns conflitos entre subtransações globais e transações locais pode mudar a

ordem de execução de transações globais. Um método de etiqueta pode ser usado para solucionar este problema. Todas as subtransações globais são forçadas a obter primeiro uma etiqueta, assim, se houver conflitos adicionais causados entre eles, a ordem de suas execução é preservada.

Em conclusão, os dois ambientes de computação – múltiplos bancos de dados e computação móvel – podem ser integrados em harmonia para fornecer uma solução para os requerimentos emergentes nesta seção apresentados.

3.6 Modelo de Transação baseado em Semântica

Neste modelo proposto em [CHRYSANTHIS, 1995] o foco é o uso de informação de semântica do objeto para melhorar a autonomia da unidade móvel em modo desconectado. Esta abordagem usa a organização de objetos e semântica de aplicação para dividir objetos (banco de dados) muito grandes e complexos em fragmentos menores do mesmo tipo [SEYDIM, 1999] e [ALVARADO et al., 2001].

O modelo de Transação baseado em Semântica assume um ambiente cliente/servidor que estão dispostos nas Estações Base (MSS). Para assegurar a consistência dos dados compartilhados na presença de concorrência e falhas, são utilizadas transações de atualização e recuperação de dados entre unidades móveis e unidades fixas.

Uma grande parte de sistemas de processamento de transações pregam um pouco de conhecimento semântico para prover maior disponibilidade de dados e alcançar um alto grau de concorrência, assim como simplificar a recuperação na presença de falhas.

A semântica dos objetos depende das seguintes características:

- Semântica de operações – é relacionado aos efeitos de uma operação no estado de um objeto;
- Valores de operações de entrada/saída – se refere a ambas as direções do fluxo de informações de e para um objeto e a interpretação dos valores de entrada e saída;

- Organização do objeto – a organização abstrata de um objeto;
- Uso do objeto – como o objeto é usado e o que é feito com a informação extraída dele.

As três primeiras características do objeto são usadas para definir várias formas de comutatividade, que determinam a semântica se duas operações puderem ser executadas concorrentemente sem compromisso de serializabilidade. A última característica, o uso do objeto, define critérios de correção quanto à aplicação específica que transcendem a comutatividade e a serializabilidade, para permitir mais operações até mesmo para executar simultaneamente e de forma assíncrona.

A propriedade semântica geralmente mais utilizada é a comutação. Se duas operações comutam, seus valores de retorno são os mesmos, independente da ordem de suas execuções. Operações que comutam podem ser organizadas arbitrariamente em uma execução concorrente para efetuar a serializabilidade.

Pode-se utilizar comutatividade para aumentar o acesso concorrente e simples recuperação de dados compartilhados na unidade móvel. Para objetos armazenados no servidor de banco de dados, se todas as operações de um objeto comutam com cada outra em todos os estados, então este objeto pode ser colocado no *cache* e pode ser manipulado na unidade móvel de forma assíncrona, sem qualquer interferência do servidor de banco de dados. É exigida da unidade móvel apenas a propagação periódica para o servidor das atualizações das transações móveis localmente consolidadas (operação *commit*). Este modelo estendido à computação móvel aumenta a concorrência explorando operações de comutatividade. Esta técnica requer para o *caching* uma porção grande do banco de dados ou mantêm cópias múltiplas de muitos itens de dados. Arquitetura ilustrada na Figura 17.

Também podem ser utilizados fragmentos de objetos de dados para facilitar a transação baseada em semântica que são processadas em bancos de dados móveis. Cada fragmento de objeto de dados tem que ser colocado em *cache* independentemente e manipulados de forma assíncrona.

Segundo [ALVARADO et al., 2001], transações móveis são invocadas pela unidade móvel, e do ponto de vista do servidor de banco de dados elas são duradouras por causa de demoras de comunicação. Unidades móveis pedem fragmentos incluindo dois parâmetros: critérios de seleção e condições de consistência. Os critérios de seleção indicam os dados para ser colocados no *cache* da unidade móvel e o tamanho de fragmento exigido. As condições de consistência especificam as restrições para preservar consistência nos dados inteiros.

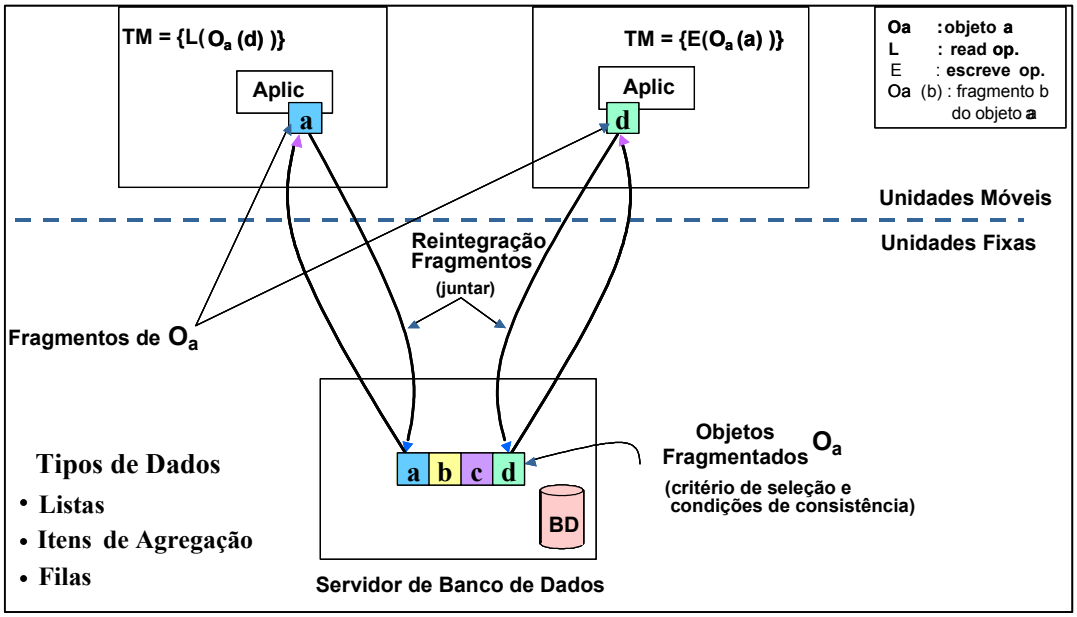

Figura 16 - Modelo Transação Baseada em Semântica.

A fragmentação de dados executada no servidor permite uma boa granulariedade para o controle de concorrência. A *cópia-mestre* exclusiva fornece cópias de fragmentos para as unidades móveis e as transações podem ser completamente executadas nestas. Um processo de reconciliação é executado pelo servidor quando a re-conexão acontecer. Este modelo pode ser usado com tipos diferentes de transação.

3.7 Modelo de Transação com Replicação em Two-Tier

Os dados são replicados em múltiplas unidades na rede, com o objetivo de aumentar a performance e também a disponibilidade dos dados. Na bibliografia estudada há tipicamente duas formas de propagar as atualizações nos dados para as réplicas [GRAY 96, LIU 99]:

- *Replicação Ansiosa (Eager)* Atualiza todas as replicas de forma sincronizada em todas as unidades, como se fosse uma transação atômica. Desta maneira satisfaz as propriedades ACID.
- *Replicação Preguiçosa (Lazy)* Faz a propagação das replicas de forma assíncrona para todas as unidades, logo após a transação de atualização. Tipicamente funciona como se fosse uma transação separada para cada unidade da rede. As atualizações são transmitidas para as outras unidades posteriormente.

Como visto em [GRAY et al., 1996], o modelo de *replicação ansioso* não é uma opção para aplicações móveis, onde a maioria dos nós está em modo desconectado. Este modelo de replicação requer níveis de **consistência rígida**, adequado para sistemas de banco de dados tradicionais.

Segundo [GRAY et al., 1996], um modelo ideal de replicação para banco de dados *wireless* precisa atingir quatro objetivos, são eles:

- Disponibilidade e escalabilidade: Fornecer altas disponibilidade e escalabilidade através da replicação, assim evitando a instabilidade;
- Mobilidade: Permitir que unidades móveis possam ler e atualizar os dados mesmo estando desconectadas da rede;
- Seriabilidade: Fornecer uma única cópia serializável de execução de transação;
- Convergência: Fornecer convergência para evitar que o sistema se comporte de uma maneira não determinística;

O comportamento instável das aplicações móveis replicadas pode ser muito sério utilizando o modelo de replicação ansioso. Para solucionar este tipo de problema, um

esquema de replicação "*Lazy Master Replication*", adequado para suprir as necessidades do ambiente de banco de dados móveis, denominado Replicação em *Two-Tiers* (duas fileiras), é proposto em [GRAY et al., 1996].

Este modelo de replicação em *Two-Tier* é um esquema de replicação que assume a existência de dois tipos de unidades, são elas:

- **Unidades Móveis** : Os nós móveis, chamados aqui por unidades móveis, estão desconectadas a maior parte do tempo. Armazenam uma réplica do banco de dados;
- **Unidades base** : Nós base, ou estações base, estão sempre conectadas. Elas também armazenam uma réplica do banco de dados.

E trabalham com dois tipos de transações: *Transações Tentativas* e *Transações Base*. As transações tentativas são originadas por uma unidade móvel, onde são executadas em modo desconectado, e mais tarde quando a conexão é re-estabelecida, com as unidades base, esta transação tentativa será transformada em uma transação base correspondente que então é re-executada.

Itens de dados replicados possuem duas versões nas unidades móveis:

- *Versão Mestre*: É a versão mais atualizada, recebida do objeto mestre. A versão deste objeto mestre é denominada versão mestre. Porém, quando as unidades estão desconectadas ou com réplicas podem ter versões mais antigas.
- *Versão Tentativa*: é criada com o valor mais recente devido a atualizações locais, este é mantido como um valor tentativo. O objeto local pode ser atualizado através de transações tentativas, e é denominado de *versão tentativa*.

Uma transação base trabalha, em modo desconectado, somente com dados mestre e produzem novos dados mestres. Estas transações envolvem no máximo uma unidade móvel conectada à rede e pode ter várias unidades fixas. A transação base gerada por uma transação tentativa pode falhar ou pode produzir resultados diferentes. A transação base tem um critério de aceitação: um teste resultante de todas as saídas tem que passar por uma ligeira diferença do resultado da transação base para ser aceitável. Enquanto

que as transações tentativas trabalham com dados locais, ou tentativos. Desta maneira elas produzem novas versões tentativas.

Se uma transação tentativa falhar, a unidade que a originou e a pessoa que gerou a transação são informados da falha.

Conforme [GRAY et al., 1996], considerando o caso desconectado, se uma unidade móvel desconectou um dia atrás, ela terá uma cópia base dos dados a partir desta data. Esta unidade gerou transações tentativas naqueles dados base e nos dados locais da cópia mestre utilizada pela unidade móvel. Essas atualizações tentativas são todas visíveis à unidade móvel, Figura 18.

Quando uma unidade móvel conecta a uma unidade fixa, esta unidade móvel:

1. Descarta suas versões tentativas desde que elas sejam atualizadas a partir dos dados mestres;
2. Envia atualizações de réplica para qualquer versão mestre, tanto para a unidade móvel quanto para a unidade fixa;
3. Envia todas suas transações tentativas para a unidade fixa para serem executadas na ordem na qual elas consolidaram na unidade móvel;
4. Aceita atualizações de réplica da unidade base (este é o padrão de replicação do mestre relaxado), e
5. Aceita notificação do sucesso ou fracasso de cada transação tentativa.

Figura 17 - Arquitetura do modelo Two-tier [GRAY et al., 1996].

A unidade fixa é uma fila das duas fileiras existentes. Quando contatado por uma unidade móvel, a unidade base:

1. Envia transações de atualização de réplica atrasadas para a unidade móvel;
2. Concorda com atualizações de transações para versões mestres da unidade móvel;
3. Aceita a lista de transações tentativas, as suas mensagens de entrada, e seus critérios de aceitação. Reprisa cada transação tentativa na ordem em que consolidou na unidade móvel. Durante este re-processamento, a transação base lê e escreve cópias de mestre do objeto que usam um modelo de execução de *master lazy*. A regra do escopo assegura que a transação base só acessa dados mestres originado do nó móvel e do nó base. Assim cópias de mestre de todos os dados na transação escopo, estão disponíveis à transação base. Se fracassar os critérios de aceitação da transação base, esta é abortada e uma mensagem diagnóstico é devolvida ao nó móvel. Se os critérios de aceitação requerem que a transação base e a tentativa tenham saídas idênticas, então transações subseqüentes que lêem resultados tentativos escritos por T também falharão. Por outro lado, a aceitação mais fraca critérios é possível;
4. Depois que a unidade base consolida uma transação base, ela propaga a réplica desta transação atualizada e envia a todas as unidades esta réplica.
5. Quando todas as transações tentativas forem re-processadas como transações bases, o estado da unidade móvel é convergido para um estado de base.

As propriedades fundamentais do esquema de replicação do modelo *two-tier* são:

1. As unidades móveis podem fazer atualizações de banco de dados tentativas;
2. Transações bases executam através de uma única-cópia serializável, assim o estado do sistema base mestre é o resultado de uma execução de serializabilidade;
3. Uma transação é consolidada quando a transação base completar;
4. Réplicas de todas as unidades conectadas convergem com o estado de sistema base;
5. Se todas as transações comutam, não há nenhuma reconciliação.

A taxa de reconciliação para transações base será zero se toda as transações comutarem. A taxa de reconciliação é dirigida pela taxa à qual as transações base falharam quanto aos critérios de aceitação. O processamento da transação base pode produzir resultados diferentes dos resultados tentativos. Isto é aceitável para algumas aplicações. Se a transação tentativa completa prosperamente e passa no teste de aceitação, então o sistema de replicação assume que tudo está bem e propagam as réplicas atualizadas.

Os usuários estão cientes que todas as atualizações são tentativas até que a transação se torna uma transação base. Se a transação base falhar, o usuário pode ter que revisar e reenviar a transação. O programador deve projetar as transações para serem comutativas e ter critérios de aceitação para descobrir se a transação tentativa concorda com os efeitos de transação bases.

Este modelo de transação móvel, assim como o modelo *Clustering*, requer que o gerenciador de transação esteja na unidade móvel para que possa executar localmente as transações, fornecer controle de concorrência, gerenciamento de *logs* e recuperação.

3.8 Modelo de Transação Prewrite

No modelo Prewrite, [MADRIA & BHARGAVA, 1998], a idéia principal é dividir a transação em partes e executar estas partes concorrentemente na unidade móvel e no servidor de banco de dados. Tendo como objetivo aumentar a disponibilidade de dados na unidade móvel introduzindo uma operação *prewrite* (pré-escrita) além da escrita padrão.

Uma operação *prewrite* permite tornar visível, antecipadamente, o valor que os dados terão depois de uma operação de escrita (*write*) ser consolidada, ou seja, um *prewrite* faz o valor de dados visível a *pré-commit* antes da consolidação da transação móvel.

A transação móvel é executada na unidade móvel, mas são feitas atualizações permanentes ao servidor de banco de dados por um gerenciador de dados (GD). *Prewrite* assegura que, delegando a responsabilidade de escrita ao banco de dados, o

processo de transação será reduzido na unidade móvel. Três operações serão executadas pela unidade móvel: *pre-reads*, *prewrites* e *precommit*. *Writes* e *reads* permanentes são feitas pelo gerenciador de dados (GD).

Cada operação de *prewrite* em uma unidade móvel deixa visível o valor da transação que eventualmente será escrita no banco de dados da estação base. Uma vez que os objetos de dados requeridos, são *reads* ou *pre-reads* e são computados *prewrite* na transação, um *pre-commit* é executado na unidade móvel.

Uma transação requer um *read* e *pre-read* de todos os objetos de dados pedidos antes do *pre-commit*, porque depois que uma transação libera um bloqueio para uma operação de *prewrite*, não pode adquirir um bloqueio para uma operação de *read* devido à condição de *two phase locking* (bloqueio em duas fases). Depois do *pre-commit*, as operações de *prewrites* são visíveis a outros processamentos de transações tanto na unidade móvel quanto na estação base. Os *prewrites* são controlados pelo gerente de transação (GT), considerando que a escrita física é controlada pelo gerente de dados.

Neste modelo de transação móvel, uma transação começa sua execução na unidade móvel. Quando uma transação chegar a unidade móvel, os pedidos de *read* da transação são processados na unidade móvel no caso de ter objetos de dados consistentes no *cache*. Caso contrário, a unidade móvel envia pedido de algumas *reads* (para qual a unidade móvel não tem nenhum objeto de dados consistente no cache) para a estação base. Quando uma transação de *read* chega a MSS, é analisada a transação e retorna os valores de *prewrite* em resposta. Se um *prewrite* não está disponível, os valores *write* são devolvidos. A MSS identifica que o valor de retorno é um valor de prewrite. No caso da transação precisar do final da write, tem que iniciar uma read novamente. Uma vez que todos valores pedidos são devolvidos da MSS para a unidade móvel, a transação é pre-committed na unidade móvel depois de declarar todos os valores de prewrite dos objetos de dados. A execução de uma transação *pre-committed* é então trocada da unidade móvel para a MSS completar a sua execução permanente que envolve a atualização do banco de dados.

Na MSS, a transação na verdade atualiza todos os objetos de dados para qual os valores de *prewrite* foram declarados mais cedo e foram consolidados depois disso. Assim, o valor de *prewrite* do objeto de dados é visível para a MSS e para outras unidades móveis naquela célula, antes do objeto de dados ser atualizado na MSS [MADRIA & BHARGAVA, 1998].
São armazenados *Prewrites* no workspace privado da transação na unidade móvel e uma vez que a transação é *pre-committed*, estes valores são movidos para a MSS.

3.9 Análise Comparativa das Propriedades ACID

Apartir dos modelos apresentados, uma análise foi feita para saber como as transações móveis lidam com as propriedades ACID e como elas são executadas. Os modelos de Transação Canguru e MDSTPM não estão incluídos porque ambos não propuseram novas soluções com respeito às propriedades ACID.

3.9.1 Atomicidade

Com exceção do modelo Reporting e do baseado em semântica, a validação de uma transação é feita em dois passos. O primeiro passo é realizado na unidade móvel (*commit local*) e o segundo passo (*commit*) é na estação base/servidor do banco de dados; Os modelos Clustering, Replicação em Two-Tier, Pro-motion e Prewrite executam *commit* local, cada um com características específicas:

- Clustering e Replicação Two-Tier, fazem commit local somente no modo desconectado, usando tipos de transações especiais. No modo conectado um protocolo de commit atômico é usado (ex., *commit* de duas fases) e inclui a participação de diversos *clusters/hosts*.
- No modelo Pro-Motion e Prewrite não é diferenciado o modo conectado do modo desconectado. O *commit* local é executado usando um protocolo de *commit* atômico.

No segundo passo do processo de validação, transações finalizadas localmente, executam *commit* para fazer as atualizações permanentes no servidor de banco de dados.

- No **Pro-Motion**, os *compacts* envolvidos nas transações finalizadas localmente são verificados. Se alguns dos *compacts* não são mais válidos, então as transações móveis são abortadas e um procedimento de contingência (anexado a cada *commit* local) é executado para obter atomicidade semântica.Neste modelo podem ocorrer *aborts* em cascata. A consolidação da transação, *commit*, pode envolver mecanismos de reconciliação ou re-execução de transações.
- Na **replicação em Two-Tier**, se as transações base (quando elas são a re-execução de transações base) falharem, mesmo levando em consideração os critérios de aceitação (anexado em cada transação tentativa), as transações tentativas serão abortadas.
- Reconciliação em ***Clustering*** é feito sintaticamente onde transações fracas são abortadas ou retornadas se suas escritas fracas conflitam com as transações rígidas.
- No **Prewrite** nem reconciliação nem re-execução são feitos. Pelo algoritmo de processamento da transação e protocolo de bloqueio, este modelo assegura que as transações consolidadas localmente, irão fazer o *commit* no servidor do banco de dados.

A abordagem é diferente no modelo *Reporting*, onde cada subtransação é atômica mas não fornece atomicidade da transação móvel global. Exceto para sub-transações não-comensatórias, transações compensatórias podem ser associadas a sub-transações, desta forma a atomicidade é garantida.

No modelo de transação baseado em semântica, as transações são consideradas transações de longa duração. Como as unidades móveis são responsáveis pelo *commit* local das transações, seria possível suportar transações atômicas e não atômicas.

Conceitualmente, os modelos baseados em semântica, Pro-Motion, Prewrite e Reporting consideram suas transações como sendo de Longa Duração. Se estas transações são executadas em sistemas de múltiplos bancos de dados, a atomicidade global depende da

autonomia de cada sistema de banco de dados. Se alguns SGBD's não podem participar de um protocolo de *commit* atômico, então se torna difícil garantir a atomicidade.

Abortos em cascata podem ocorrer no modelo *clustering*, replicação em *Two-tier* e *Pro-motion*. Não obstante, as transações que fazem *commit* local modificam dados locais, conseqüentemente, somente abortos de transações locais são provocados. Além disso, esses abortos dizem respeito somente a transações fracas e tentativas porque resultados locais são exclusivamente disponíveis para esses tipos de transações.

A Tabela 3 mostra o processo de validação dos modelos comparados. Vale a pena salientar o fato que as transações móveis fazem a validação em dois passos, onde o *commit local* é feito na unidade móvel e o *commit* é feito na MSS (estação base) servidora do banco de dados.

Modelo	**Processo de Validação**	
	Primeiro Passo – na UM	Segundo Passo – servidor do BD
Clustering	Modo Desconectado: *commit local* de transações Fracas. Modo Conectado: c*ommit* de duas fases para transações rígidas	Commit envolve reconciliação em "*aborts*" e "rollback" em solução de conflitos
Replicação em Two-Tier	Modo Desconectado: *commit local* de transações tentativas. Modo Conctado: Protocolo de *commit* atômico para transações base.	Transações Tentativas são re-executadas levando em consideração seu critério de aceitação.
Pro-Motion	*Commit local* de todas as transações locais	Um processo de sincronização verifica os *compacts* envolvidos nas transações locais. Em caso de conflitos, transações locais são abortadas e procedimentos de contingência são executados.
Prewrite	*Commit local* de todas as transações locais	Atualizações locais são feitas permanentemente por operações *write* (escrita)
Baseados em Semântica	*Commit local*	Reintegração de atualizações. Como os fragmentos são cópias exclusivas e eles possuem condições de consistência anexada, não existem conflitos na reitntegração
Reporting	Toas as subtransações são atômicas e elas são capazes de consolidar independentemente de suas transações raízes.	

Tabela 3 - Sumarização de Processo de Validação

3.9.2 Consistência

Os modelos *clustering* e *replicação em Two-Tier* mantém a consistência dos dados replicados com duas versões. Ambas as versões estão localizadas na unidade móvel, uma delas (fraca/tentativa) é usada para suportar a evolução dos dados no modo desconectado. A segunda versão (rígida/mestre) deve ser sempre consistente mas algumas vezes poderá conter versões mais antigas (quando estiver no modo desconectado). Consistência em versões rígidas/mestre é preservada usando uma cópia mestre. Algumas particulariedades:

- Em *clustering*, informação semântica é usada para especificar o nível de inconsistência de versões fracas. Este nível pode ser limitado restringindo o número de *commits* locais, o número de transações que podem operar em cópias inconsistentes, o número de cópias que podem divergir, etc. Existe também uma função *h* que controla este nível através de projeções de operações rígidas em versões fracas. Uma consistência total é obtida através da junção de diferentes cópias dos mesmos dados localizados em diferentes clusters (reconciliação).
- Em replicação Two-Tier, versões tentativas são descartadas na reconexão desde que elas estejam completamente desvinculadas das versões mestre.

Nos modelos Pro-motion e baseado em semântica exploram informação semântica para construir os *compacts* e fragmentos respectivamente:

- Para o pro-motion o *compact* representa um acordo entre o servidor de banco de dados e a unidade móvel. O gerenciador de *compact* e o servidor de banco de dados encapsulam nos compacts: dados, métodos específicos de tipos, informação de estado, regras de consistência, e obrigações. Se o agente *compact* e o gerenciador de *compact* respeitarem todas essas condições, o uso de *compacts* não afetará na consistência do banco de dados. O projetista do *compact* pode determinar o critério de correção e métodos de controle de concorrência deste objeto.
- Em baseados em semântica, para preservar a consistência, objetos devem ter cuidado no suporte ao particionamento (para fazer os fragmentos) e as operações de junção (para fazer a reconciliaçãodos fragmentos). Uma outra restrição para preservar a consistência é fornecer *condições de consistência* (fornecido pelas

aplicações) no objeto inteiro. Estas condições incluem operações permissíveis, restrições de seus valores de entrada e condições no estado do objeto.

No modelo *reporting*, novas maneiras de atingir consistência não foram propostas, mas sub-transações podem ser relacionadas às transações compensatórias (exceto para não compensatórias) com o objetivo de manter a consistência semântica em caso de abortos.

Prewrite assegura que o algoritmo do processamento de transações com o protocolo baseado em bloqueio, produzem somente histórico serializável.
É importante ressaltar que informação semântica de objetos é essencial para garantir a consistência em aplicações móveis. Todos os modelos analizados exploram objetos semânticos em diferente formas. *Clustering* define níveis de inconsistência baseados nas aplicações semânticas. *Replicação em Two-Tier* gerencia um critério de aceitação entre transações base e transações tentativas. Pro-motion usa informação semântica para construir os *compacts* e modelo *baseado em semântica* para particionar os fragmentos. *Reporting* faz delegações baseadas em requerimentos semânticos, e Prewrite define semanticamente variantes de dados idênticos (prewrite/objetos write)

Modelo	**Conceitos de**	**Uso da informação semântica**
Clustering	Duas versões de Dados: Rígida (uma cópia serializável), fraca (níveis de incinsistência, evolução dos dados no modo desconectado)	Definição de uma função *h* e níveis de inconsistência
Replicação em Two-Tier	Duas versões de Dados: Master (uma cópia serializável), Tentativa (dados locais evolução no modo desconectado)	Critério de Aceitação
Promotion	*Compacts* incluem métodos específicos de tipos, regras de consistência e obrigações	Construção de compacts e procedimentos de contingência
Reporting	Abordagem Multitransações	Transações Compensatórias e Delegação
Baseados em Semântica	Fragmentação de Objetos (condições de consistência e operações de particionamento/junção)	Fragmentação
Prewrite	A seriabilidade é baseada na ordem de *commit* local das transações móveis	Definições de variantes de dados (*prewrite/write*)

Tabela 4 - Sumarização das Propriedades de Consistência

A Tabela 4 sumariza os principais conceitos usados para preservar a consistência. E também enfatiza a importância da informação semântica para oferecer mais flexibilidade no suporte a consistência.

O próximo tópico referente à análise dos modelos com respeito à propriedade de *isolamento* está fortemente relacionado com a propriedade de *consistência*, porque a *execução de uma transação em isolamento preserva a consistência,* [ALVARADO et al., 2001].

3.9.3 Isolamento

A propriedade de Isolação das transações assegura que uma transação não irá interferir na execução de uma outra transação. Este propriedade é geralmente obtida através de mecanismos de controle de concorrência. Assim como a propriedade de atomicidade, o isolamento é necessário para que a consistência dos dados seja preservada [KUMAR & DUNHAM, 1998].

Os modelos *Clustering*, replicação em *Two-Tier*, *Pro-motion* e baseado em semântica permitem visibilidade de resultados consolidados localmente (*commit* local) para transações locais da mesma unidade móvel. Por outro lado, *Prewrite* torna públicos os resultados de transações locais para todos os hosts/unidades.

No modelo reporting, visibilidade é permitida nas *transações atômicas*, *reporting* e *co-transações*, mas não em transações não-compensatórias. Uma **transação atômica** pode fazer o *commit* mesmo antes que suas transações originárias não tenham feito, e estas atualizações no banco de dados se tornam visíveis para outras transações.
Nas transações *reporting* e **co-transações** o objetivo é permitir visibilidade de resultados parciais enquanto em execução.

Tendo *Pro-motion* e *reporting* como transações aninhadas abertas, a isolação global não é respeitada desde que sub-transações sejam executadas isoladamente. Depois do processo de sincronização, o *Pro-motion* particiona sua transação de longa duração. Todas as operações que foram sincronizadas com sucesso formarão uma transação separada que é consolidada (é feito o *commit*) no servidor do banco de dados. Os resultados deste particionamento serão visíveis para todos os ambientes de banco de dados.

Para gerenciar a propriedade de isolamento (restrição de visibilidade) *Clustering* e *Prewrite* propuseram novas tabelas de soluções de conflito.

- *Clustering* usa bloqueio rígido de duas fases e propõe quatro tipos de bloqueio que correspondem a operações rígidas e fracas: leitura fraca (WR), escrita fraca (WW), leitura rígida (SR) e escrita rígida (SW). Quatro tabelas de conflito para compatibilidade de bloqueio são propostas. A *função desenvolvedora* ***h*** utiliza tabelas de conflito para refletir operações rígidas em versões fracas dependendo dos requerimentos de consistência das aplicações. Por exemplo, consistência rígida requer a tradução de uma escrita rígida em um objeto em escrita rígida em todas as suas cópias (rígidas ou fracas). Conseqüentemente, uma escrita rígida não é compatível com qualquer outro bloqueio. Transações fracas liberam seus bloqueios no *commit* local e transações rígidas no *commit*.
- Assim como *Clustering, Prewrite* usa um protocolo de bloqueio de duas fases e a tabela de operação de conflitos inclui operações de *pre-read* (pré-leitura) e *prewrite* (PR, PW, R, W). Como os bloqueios de *pre-read* e *prewrite (pré-escrita)* são gerenciados no nível do gerente de transações e os bloqueios de *escrita* e *leitura* no gerente de dados, não existem conflitos entre bloqueios *prewrite*/*pre-read* e *writes*/*read*. Para tornar as operações de prewrite permanentes o bloqueio de prewrite deve ser convertido em um bloqueio de escrita para que então o gerenciador de dados possa escrever e fazer o *commit* da transação móvel. Bloqueios de *preread* são liberados no *commit* local enquanto bloqueios *prewrite/write* e *reads* são liberados no *commit*.

No *Pro-motion* o projetista do objeto *compact* pode determinar os métodos de critérios de correção e controle de concorrência por *compact*, neste modelo é proposta uma escala de dez níveis. Estes níveis são caracterizados baseados em níveis de isolamento definidos no padrão SQL ANSI [WALBORN & CHRYSANTHIS, 1999]. O nível *nove*, por exemplo, representa uma execução serial de transações e o nível *oito* uma execução serializável. Cada nível abaixo representa um menor grau de isolamento. No nível *zero* não há nenhuma garantia de isolação. Devido ao arbitrário uso de níveis de isolamento é possível que ocorram inconsistências, Pro-Motion propõe algumas regras:

- Transações impõem nível mínimo para operações de escrita e leitura;
- Cada operação é associada a um nível;
- Nenhum dos níveis de operação de escrita é menor que o nível de escrita da transação corrente;
- Nenhum dos níveis de operação de leitura é menor que o nível de leitura da transação;
- O menor nível de qualquer operação de leitura é maior ou igual que o maior nível requerido por qualquer operação de escrita.

No modelo baseado em semântica, para assegurar a serialização, transações locais tem acesso aos fragmentos no *cache* por protocolos convencionais de controle de concorrência, no caso **bloqueio de duas fases**.

Modelo	**Visibilidade**	**Protocolo de Controle de Concorrência**
Clustering	Resultados de transações consolidadas localmente são visíveis para transações fracas (locais) na mesma UM	Protocolo de bloqueio em duas fases e novos tipos de bloqueios são propostos
Replicação em Two-Tier	Resultados de transações consolidadas localmente são visíveis para transações tentativas locais na mesma UM	Mecanismos de Bloqueios.
Promotion	Resultados de transações consolidadas localmente são visíveis para transações tentativas locais na mesma UM	Protocolo de bloqueio em duas fases
Reporting	A visibilidade com subtransações atômicas, reporting e co-transações é permitida antes da consolidação da transação global	Protocolo de bloqueio em duas fases
Baseados em Semântica	Resultados de transações consolidadas localmente são visíveis para transações tentativas locais na mesma UM	Protocolo de bloqueio em duas fases
Prewrite	Resultados de transações consolidadas localmente são visíveis para todas as unidades	Protocolo de bloqueio em duas fases, e novos tipos de bloqueios são propostos.

Tabela 5 - Sumarização dos aspectos da propriedade de Isolamento.

Na Tabela 5, é ressaltada a importância da visibilidade no *commit* local. Com a disponibilidade dos dados a UM possui alguma autonomia, conseqüentemente, o processo local na UM não será parado quando uma desconexão ocorrer. Além disso, a tabela torna evidente que o protocolo de concorrência mais utilizado é o de **bloqueio em duas fases.**

3.9.4 Durabilidade

Os modelos *Clustering*, *Replicação em Two-Tier* e *Pro-Motion* não podem garantir a durabilidade antes da operação de commit. No modelo Pro-Motion a utilização dos objetos *compacts* fornecem alguma garantia quanto à propriedade de durabilidade, mas podem existir condições que não poderiam ser respeitadas devido às desconexões, por exemplo, no *compact* há um prazo limite (deadline) que não poderia ser alcançado, conseqüentemente a durabilidade é dificultada no processo de sincronização.

No *Reporting* , sub-transações são duráveis se as transações raízes/país executarem um *commit*. Os modelos de transação *baseado em semântica* e *Prewrite* garantem a durabilidade desde o *commit* local. O primeiro reduz a disponibilidade dos fragmentos porque ele pode manusear estes fragmentos por um período de tempo indefinido. E o segundo utiliza muitas mensagens de troca para obter bloqueios da estação base. No algoritmo do *Prewrite*, se uma transação móvel executa um *commit* local, é certo o *commit*, *Prewrite* não permite o cancelamento de uma transação consolidada localmente.

Modelo	**Durabilidade é garantida**	**Inconveniente**
Clustering	Sim, após a consolidação	Transações consolidadas localmente pode ser "rolled back" (voltada até o último ponto de *commit*) devido a conflitos de resincronização
Replicação em Two-Tier	Sim, após a consolidação	Transações consolidadas localmente pode ser "rolled back" (voltada até o último ponto de *commit*) devido a conflitos de resincronização durante a re-execução
Promotion	Sim, após a consolidação	Transações consolidadas localmente pode ser "rolled back" (voltada até o último ponto de *commit*) devido a conflitos de resincronização
Reporting	Sim, se as transações adjacentes consolidarem, as subtransações são duráveis	
Baseados em Semântica	Sim, após a consolidação local	Redução de disponibilidade de fragmentos no servidor de banco de dados
Prewrite	Sim, após a consolidação local	Muitas mensagens trocadas entre a UM e a MSS

Tabela 6 - Sumarização da propriedade de Durabilidade

A Tabela 6 mostra os momentos quando a durabilidade não é assegurada e alguns inconvenientes.

3.10 Análise Comparativa dos Modelos de Transação

Clustering assume um sistema completamente distribuído e foi desenvolvido para manter a consistência do banco de dados. O banco de dados é dividido em clusters, agrupados por dados semanticamente relacionados ou fisicamente localizados próximos. Um *cluster* pode ser distribuído em várias unidades fortemente conectadas. Quando a UM se desconecta ela própria se torna um *cluster*. Para cada objeto uma duas versões são mantidas, uma delas (versão rígida) deve ser globalmente consistente, e a outra (versão fraca) pode tolerar alguns níveis de inconsistência, mas deve ser localmente consistente. As transações são classificadas como *rígidas* ou *fracas*. Transações fracas acessam somente versões fracas enquanto as rígidas acessam versões rígidas.

Transações rígidas são executadas quando as unidades estão fortemente conectadas e transações fracas quando a UM está desconectada. Dois tipos de operações são introduzidos: leituras fracas (*weak reads*) e escritas fracas (*weak writes*). Transações rígidas contém operações de leituras e escritas normais, enquanto que transações fracas contém operações fracas. Quando a reconexão é possível, um processo de sincronização, executado no servidor de banco de dados, permite que o banco de dados fique globalmente consistente.

T*wo-tier* considera um sistema móvel onde as UMs estão ocasionalmente conectadas. Uma versão mestre, *master*, para cada item de dado e muitas réplicas deste dado (cópia) existem. Dois tipos de transação são suportados: transações base e tentativas. Transações Base são executadas acessando a **versão mestre**, enquanto que transações tentativas são executadas acessando **versões tentativas** (cópias locais). Transações tentativas podem desenvolver atualizações na UM em modo desconectado. Quando a conexão é estabilizada as transações tentativas são re-executadas como transações base para a obtenção da consistência global do banco de dados. Esta re-execução é a forma de tornar persistentes as atualizações locais.

Pro-motion o processamento das transações suportam o modo desconectado. Os *compacts* são introduzidos para permitir a execução local nas UMs. Informações necessárias para gerenciar os *compacts* são encapsuladas nele. Para melhorar a autonomia e aumentar a concorrência, semântica de objetos são usados na construção dos *compacts* sempre que possível. *Compacts* são unidades básicas de armazenamento e controle. A construção dos *compacts* é de responsabilidade do GT, *Gerenciador de Compact,* que fica no servidor do banco de dados. O gerenciamento dos *compacts* fica a cargo do GT, de um AG, *agente compact*, na UM, e de um *gerente de mobilidade* que fica na MSS. Em cada UM, o *agente compact* é responsável pelo gerenciamento de *cachê,* processamento da transação, controle de concorrência, arquivos de *logs* e recuperação. O gerente de mobilidade está na troca de transmissões entre os agentes. As transações na UM são executadas localmente mesmo estando em modo conectado. Um processo de sincronização é executado pelo agente *compact* juntamente com o gerenciador de *compact* na reconexão. Este processo verifica os *compacts* modificados por transações consolidadas localmente. Se os *compacts* preservam a consistência global, então a consolidação global é executada.

Reporting uma transação móvel é estruturada como um conjunto de transações, em que algumas das quais são executadas na UM. Analisa transações aninhadas e transações aberto aninhadas mostrando suas limitações para o ambiente móvel. Este modelo considera um ambiente de bancos de dados móveis como um especial sistema de multibanco de dados com requisições específicas, onde transações na UM são consideradas como um conjunto de subtransações. Este modelo propõe uma transação aberto aninhada que suporta atomicidade, transações não-compensáveis e dois tipos adicionais: *reporting* e *co-transações*. Enquanto em execução, transações podem compartilhar seus resultados parciais e mantém parcialmente o estado de uma subtransação (executada na UM) em uma MSS.

Baseados em Semântica Transações móveis são invocadas pelas UMs, e o ponto de vista dos servidores de banco de dados é que elas são transações de longa duração (ou longa vida) devido aos atrasos de comunicação. A estrutura da transação não foi proposta podendo ser utilizado diferentes tipos. A unidade móvel requisita fragmentos

incluindo dois parâmetros: *critério de seleção* e *condições de consistência*. O critério de seleção indica os dados que precisam ser armazenados na UM e o tamanho dos fragmentos requeridos. As condições de consistência especificam as restrições para a preservação da consistência dos dados.

Prewrite a idéia principal é dividir a execução da transação entre UM e servidor do banco de dados. O Gerenciador de transações na UM executa as transações, mas atualizações permanentes são executadas no servidor de banco de dados pelo gerenciador de dados (GD). Este modelo assegura que, pela delegação da responsabilidade de operações de *write* ao banco de dados, o processamento da transação é reduzido na UM. Três operações, *pre-reads, prewrites* e *precommit*, que serão executadas pelo GT são propostas. As operações permanentes, *reads* e *writes*, são executadas pelo GD. A execução das transações é dividida em duas partes, primeira, o GT requisita à MSS os bloqueios necessários. A MSS adquire estes bloqueios do GD. Quando o GT termina a execução da transação por um *commit* local, as operações de *prewrite* são enviadas para a MSS. Na segunda parte, o GD torna os *prewrites* permanentes no banco de dados e consolida a transação efetivamente. Este modelo considera a transação como de longa duração e a sua implementação pode ser feita com transações aninhadas ou transações particionadas.

Transação Canguru propõe um modelo de transação móvel que focaliza no movimento da unidade móvel durante a execução das transações. As transações móveis são inicializadas na UM e são completamente executadas na rede fixa. O TC propõe a implementação de um *Agente de Acesso a Dados*, o AAD, no topo do gerenciador de transações globais. Este agente é situado na MSS e irá gerenciar as transações móveis e o movimento da UM. Quando uma unidade móvel muda de uma célula para outra (de MSS para outra MSS), faz *handoff,* a coordenação da transação também se move. Esta mobilidade é capturada pela divisão da transação original em duas transações, chamadas de transações filhotes, *joeys,* que ficam uma em cada célula. Assim, se a UM salta da MSS-1 para a MSS-2, a MSS-1 coordenará apenas as operações que foram executadas durante a estada da UM na célula da MSS-1.

MDSTPM Cada MDSTPM é responsável pela coordenação de transações globais. Quando uma unidade móvel envia uma transação global, ela poderá se desconectar e desenvolver outras tarefas sem ter que esperar pela consolidação da transação móvel. Utiliza o mecanismo Message Queuing Facility para trocar mensagens entre a UM e a MSS.

A principal idéia do mecanismo de Message Queuing Facility (MQF) é a troca de mensagem de forma assíncrona. Estas mensagens são baseadas nos seguintes tipos: request, acknowledgment, e information [YEO & ZASLAVSKY, 1994]. Com este mecanismo a unidade móvel pode submeter transações globais e passar para o modo desconectado. Em uma unidade móvel e sua estação coordenadora existem tabelas e logs que registaram o estado completo da UM também como transações globais (fila de mensagens, fila de transações, log globais, tabela de transação global, tabela de status do site). Assim, a qualquer momento, uma unidade móvel poderá requisitar informações sobre suas transações globais. A coordenação das transações móveis neste modelo é feita de forma centralizada.

Na Tabela 7 foi feita uma análise comparativa entre os modelos de transações móveis, sumarizando aos seus respectivos modelos de execução.

Modelo	Transação	Requisição	Execução na UM	Execução na rede fixa
Clustering	Transações rígidas e fracas	UM	Transações fracas e *commit local* em modo desconectado. Transações rígidas participam da execução no modo conectado	Transações rígidas e *commit* de transações fracas (sincronização, atualizações permanentes).
Relicação Two-tier	Transações Base e Tentativas	UM	Transações tentativas no modo desconectado. Participação na execução de transações Base no modo conectado	Transações Base
Pro-motion	Transações Aninhadas Particionadas "Split" e Transações de TLD (Transações de Longa Duração)	UM	O agente *compact* executa completamente a transação móvel e faz *commit* local	O gerenciador do *compact* está na carga da construção do *compact, commit* de transações consolidadas localmente (sincronização, atualizações permanentes).
Reporting	Transações Aninhadas	UM / Estação Base	Subtransações e transações globais	Transações Globais e Subtransações

	Abertas, atômicas, não-compensável, "reporting" e co-transações			
Baseadas em semântica	TLD	UM	Transações locais e *commit* local	Em resposta as requisições da unidade móvel, a fragmentação dos objetos é feita pelo servidor do banco de dados e também reintegração de atualizações (merge).
Prewrite	TLD (aninhadas, transações split)	UM	Transações locais e *commit* local	Gerenciamento de bloqueios e *commit* de transações finalizadas localmente (operações de escrita)
Canguru	Transações "Split" e aninhadas	UM		Coordenação e execução de transações completas.
MDSTPM	Multi-transações e transações locais	UM	Transações Locais	Coordenação e execução de multi-transações.

Tabela 7 - Tabela de Comparação entre os Modelos de Transações Móveis

4 Proposta de uma Taxonomia Diferenciada

De acordo com as necessidades dos modelos de transações móveis (apresentadas no capítulo anterior), foi feita uma proposta de taxonomia para os mecanismos de controle de concorrência em bancos de dados *wireless*.

Abordamos, ainda, os mecanismos de controle de concorrência para disseminação dos dados em um ambiente móvel através de *broadcast*, onde o servidor de broadcast continuamente envia itens de dados atualizados para os clientes móveis através de uma rede *wireless*. Entretanto se as atualizações nos servidores de banco de dados são feitas concorrentemente, as transações móveis podem observar valores de dados inconsistentes, isto é, o escalonamento de execução resultante, das transações de atualizações e transações móveis, pode ser não-serializáveis.

Há inúmeras maneiras para classificar as abordagens de controle de concorrência. Um exemplo de critério de classificação é o modo de distribuição do banco de dados. Alguns algoritmos propostos na literatura requerem um banco de dados completamente replicado, enquanto que outros podem operar em banco de dados parcialmente replicados ou particionados. Os algoritmos de controle de concorrência também podem ser classificados de acordo com a topologia da rede, tais como rede em estrela, rede em anel.

Os mecanismos de controle de concorrência, na taxonomia aqui proposta, são classificados em termos de primitivas de sincronização, como [ÖZSU & VALDURIEZ, 1999] propôs para controle de concorrência em banco de dados distribuídos.

Os algoritmos são divididos em dois grandes grupos principais: aqueles baseados em acesso mutuamente exclusivo (bloqueios) e os que objetivam ordenar a execução das transações (*Timestamp*) de acordo com um conjunto de regras (protocolos). Além destes um outro grupo tem sido largamente utilizado em controle de concorrência para banco de dados móveis, o grupo *híbrido*, que juntamente com bloqueio utiliza *timestamps*.

Estas primitivas podem ser usadas nos algoritmos em diferentes pontos de vista, a *visão pessimista,* onde muitas transações irão conflitar, ou a *visão otimista* onde menos transações irão conflitar. Além destas duas visões, para suportar as características do ambiente móvel foi proposta a *visão híbrida*. Assim agrupamos os mecanismos de controle de concorrência em três grandes classes: métodos de controle de concorrência pessimistas, métodos de controle de concorrência otimista e em métodos de controle de concorrência híbrido.

O grupo pessimista é composto pelos algoritmos baseados em *Bloqueio,* algoritmos baseados em ordenação *Timestamp* e pelos algoritmos *Híbridos* (utilização de bloqueio com atribuição de *timestamp*)**.** O grupo otimista é composto pelos algoritmos baseados em *Bloqueio* e *Timestamp*. Já o grupo *Híbrido*, integra as abordagens pessimista e otimista.

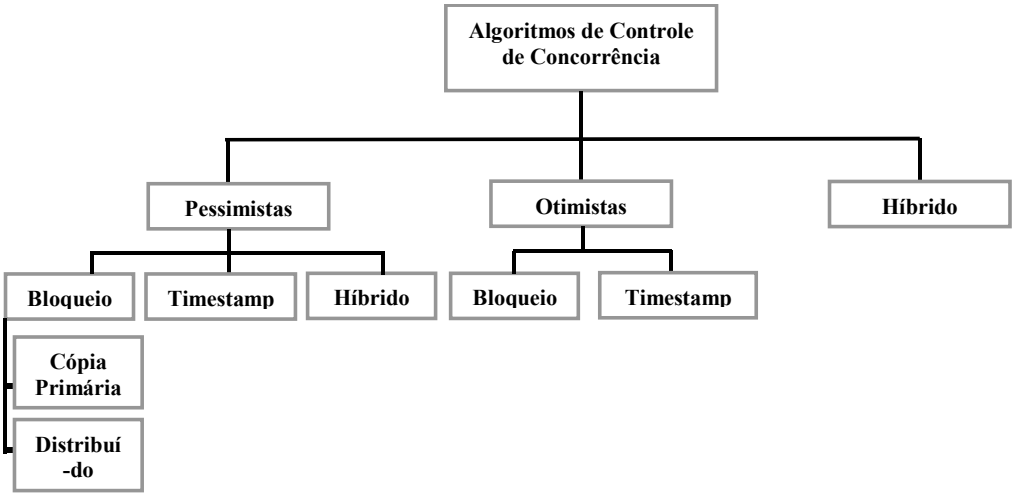

Figura 18 – Classificação dos Algoritmos de Controle de Concorrência para BDW. Modificada de **[ÖZSU & VALDURIEZ, 1999]** .

Na abordagem baseada em bloqueios, a sincronização das transações é obtida través do emprego de bloqueios físicos ou lógicos. Esta abordagem pode ser subdividia em de acordo com os locais onde estes bloqueios são executados:

- *Bloqueio de cópia primária*: Os itens de dados são replicados e um destes itens é designado a ser a cópia primária, ou seja, a cópia considerada consistente, onde as atualizações serão feitas. Esta cópia deve ser bloqueada para o propósito de acessar essa unidade de bloqueio específica. Caso o banco de dados não seja replicado, os mecanismos de bloqueio de cópia primária distribuirão a responsabilidade pelo gerenciamento do bloqueio entre os diversos *sites*.
- *Bloqueio descentralizado:* A tarefa do gerenciamento de bloqueios é compartilhada por todos os sites de uma rede. Neste caso, a execução de uma transação envolve a participação e coordenação de escalonadores em mais de um site. Cada escalonador local é responsável pelas unidades de bloqueio local.

4.1 Abordagem Pessimista

Na abordagem pessimista os algoritmos são classificados em: baseados em Bloqueio e baseados em ordenação de execução por *Timestamp.* Sua utilização é indicada quando a unidade móvel está em modo conectado, ou seja, ambiente onde podem ocorrer muitos conflitos entre as transações.

Nos *algoritmos baseados em bloqueio* a sincronização é obtida pelo emprego de bloqueios físicos ou lógicos em alguma porção ou granulo do banco de dados. O tamanho destas porções (geralmente chamada de granulariedade de bloqueio) não será abordado, sendo referenciado sempre como uma unidade de bloqueio.

O controle de concorrência, na maioria dos sistemas móveis, baseia-se em bloqueios, da mesma maneira como na maioria dos sistemas distribuídos e também não distribuídos. No sistema móvel, entretanto, as solicitações de teste, requisições e liberação de bloqueios tornam-se mensagens, e mensagens adicionais significam sobrecarga no sistema, que geralmente opera com baixa largura de banda.

Os *algoritmos baseados em ordenação de execução ou TimeStamp* consistem em associar a cada transação T_i um único *timestamp* fixo, $TS(T_i)$, em sua inicialização [ÖZSU & VALDURIEZ, 1999] e [SILBERSCHATZ et al., 1999], podendo ser usado o próprio relógio do sistema.

4.1.1 Mecanismos de Controle de Concorrência baseados em Bloqueio

A idéia básica do controle de concorrência baseado em bloqueios é assegurar que os dados que são compartilhados por operações conflitantes sejam acessados somente por uma operação de cada vez [ÖZSU & VALDURIEZ, 1999].

Um meio de garantir a serialização é obrigar que o acesso aos itens de dados seja feito de maneira mutuamente exclusiva; isto é, enquanto uma transação acessa a um item de dados, nenhuma outra transação pode modificá-lo . O método mais usado para sua implementação é permitir o acesso a um item de dados somente se ele estiver bloqueado [SILBERSCHATZ et al., 1999].

Um bloqueio é adquirido por uma transação antes de ser acessado e é reinicializado depois de ser utilizado. Uma unidade de bloqueio não pode ser acessada por uma operação se já está bloqueada. Assim uma requisição de bloqueio feita por uma transação é obtida somente se a unidade de bloqueio associada não estiver sendo utilizada por outra transação.

Nos sistemas baseados em bloqueio, o escalonador das operações das transações é chamado de **Gerenciador de Bloqueios (GB)**. O Gerenciador de Transações (GT) passa para o GB as operações de banco de dados (leituras/escritas) e informações associadas (como qual item deverá ser acessado e o identificador da transação que lida com esta operação).

O GB verifica se a unidade de bloqueio que contém o dado requerido já está bloqueada. Se ela estiver bloqueada, e se o modo de bloqueio existente for incompatível com a transação atual, a operação corrente é colocada em modo de espera. Caso contrário, o

bloqueio para a operação corrente é concedido e a operação do banco de dados é passada para o **Processador de Dados (PD)** para acesso ao banco de dados, como mostra a Figura 20. O gerenciador de transações é então informado dos resultados da operação. O término da transação resulta nas liberações dos seus bloqueios e na inicialização de outra transação que já deve estar em modo de espera para o acesso ao mesmo item de dados.

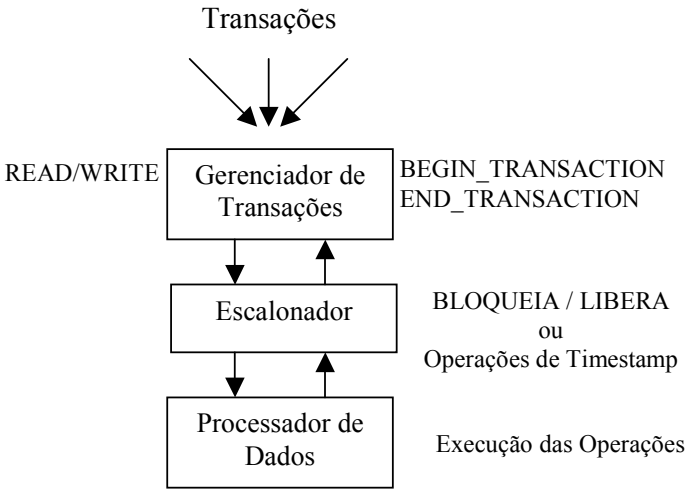

Figura 20 - Estrutura Geral de Gerenciadores para manipular Transações.

No entanto esse modo de execução permite a ocorrência de muitos *deadlocks*, pois este mecanismo libera os bloqueios de uma transação logo que os comandos de banco de dados associados (leituras e escritas) são executados, e esta unidade de bloqueio (chamamos *x*) logo poderá ser acessada. Entretanto, a própria transação estará bloqueando outros itens (por exemplo, *y*), depois libera seu bloqueio em *x*. Apesar de parecer uma vantagem, do ponto de vista do aumento de concorrência, este método permite que uma transação interfira em outra, resultando em perda total de isolação e atomicidade.

Esses problemas causam a necessidade de coordenar as operações de bloqueio e de liberação de bloqueio, implementando um **Método de Bloqueio em Duas Fases.** Os algoritmos de controle de concorrência baseados no bloqueio em duas fases, executam transações em duas fases, a **Fase de Crescimento** e a **Fase de Contração**. A **Fase de**

Crescimento é onde a transação obtém todos os bloqueios necessários e acessa os itens de dados requeridos. Na **Fase de Contração** todos os bloqueios requeridos são liberados [ÖZSU & VALDURIEZ, 1999]. Isto permite que outras transações que estejam esperando o acesso a estes itens de dados, logo consigam os seus bloqueios, aumentando assim o nível de concorrência.

Para transações móveis o algoritmo de controle de concorrência baseado em bloqueios utilizado na abordagem pessimista é o algoritmo **B2F-D,** algoritmo de *bloqueio em duas fases distribuído*.

4.1.1.2 Bloqueio em Duas Fases com Cópia Primária

O mecanismo de bloqueio em duas fases com cópia primária é considerado uma extensão do bloqueio em duas fases centralizado para bancos de dados distribuídos. Esta extensão serve para solucionar problemas encontrados neste algoritmo, como o gargalo formado pela centralização, que o torna menos confiável devido, por exemplo, a um único ponto de falha.

Basicamente este mecanismo implementa gerenciadores de bloqueio em vários *sites* e faz com que cada gerenciador de bloqueio seja responsável pelo gerenciamento de um dado conjunto de unidades de bloqueio. Os gerenciadores de transações enviam suas requisições de bloqueio e desbloqueio para os gerenciadores de bloqueio. Assim o mecanismo trata uma cópia de cada item de dados como cópia primária. O funcionamento do mecanismo é ilustrado esquematicamente na Figura 21.

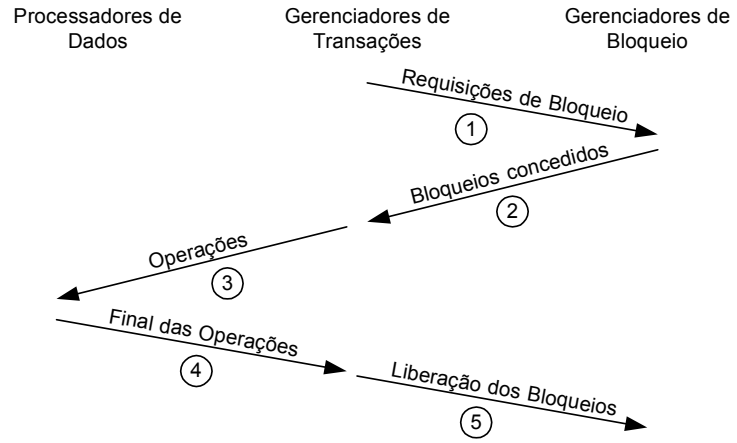

Figura 21 - Estrutura de Comunicação do B2F-CP

4.1.1.3 Bloqueio em Duas Fases Distribuído

O mecanismo **B2F-D** (bloqueio em duas fases distribuído) conta com a disponibilidade de vários gerenciadores de bloqueios. A comunicação entre os *sites* participantes que executam o protocolo de B2FD é apresentada na Figura 22.

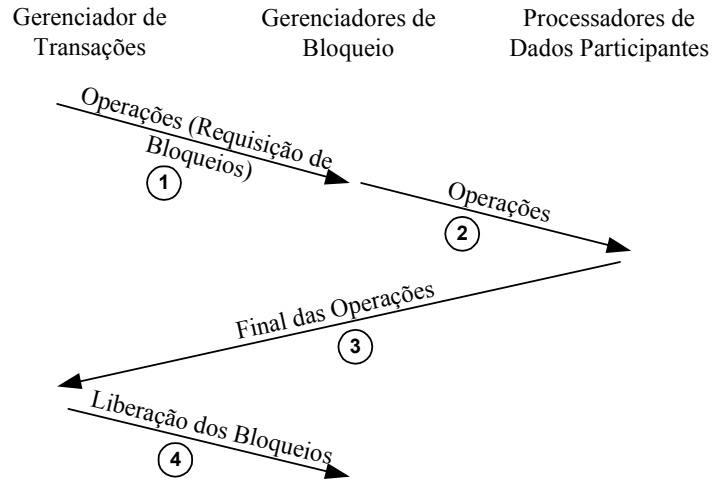

Figura 22 - Estrutura de Comunicação do B2F-D.

As mensagens são enviadas do GT para todos os *sites* gerenciadores de bloqueio participantes do mecanismo de B2F-D, e os GBs enviam as mensagens diretamente para os processadores de dados, ao contrário do algoritmo B2F-C (bloqueio de duas fases centralizado) [ÖZSU & VALDURIEZ, 1999].

A Figura 23 apresenta um exemplo de utilização de bloqueio em duas fases distribuído no qual uma unidade móvel migra para outras células enquanto a transação ainda não está consolidada.

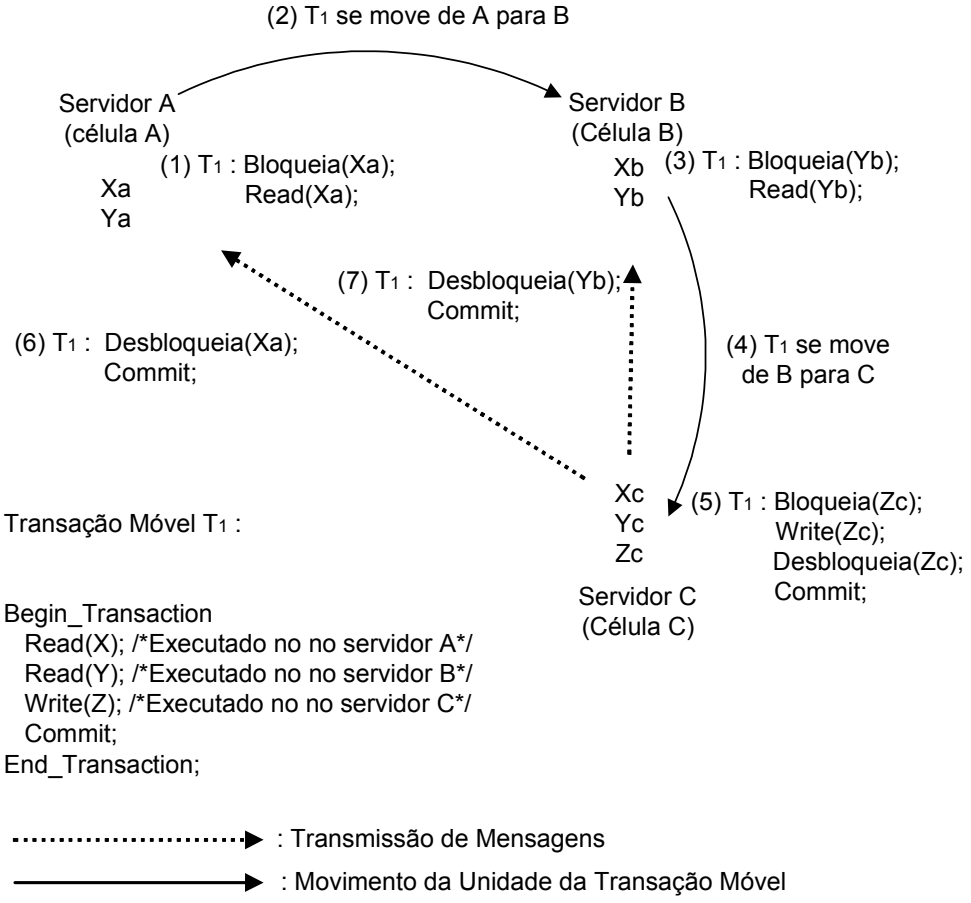

Figura 23 - Transação Móvel usando o algoritmo B2F [JING et al., 1995].

Considere que no sistema de banco de dados *wireless* apresentado na Figura 23, os dados X e Y são replicados nas unidades A, B e C e o dado Z tem uma cópia na unidade C. A unidade da transação móvel T1 requisita uma operação de leitura no item X

(read(X)), na unidade A e então se desloca para a unidade B. Solicita uma operação de leitura no item Y (read(Y)) e se move para a unidade C, onde requisita uma escrita no item Z (write(Z)) e uma operação de consolidação desta transação (*commit*). Para ler cada item, o coordenador em cada unidade requisita imediatamente um bloqueio de leitura e uma operação de leitura no servidor local. Em tempo de *commit,* entretanto, o coordenador da unidade C precisa enviar duas mensagens de *commit* (ou desbloqueio) para os servidores A e B para a liberação dos bloqueios de leitura.

Este esquema tem a vantagem de uma implementação simples, requerendo apenas duas transferências de mensagem para manipular requisições de bloqueio e uma transferência de mensagem para manipular requisições de desbloqueio.

As mensagens de requisição de bloqueio são enviadas para todas as unidades. As operações não são enviadas para o processador de dados pelo gerenciador de transação, mas por todos os gerenciadores de bloqueio participantes.

4.1.2 Mecanismos de Controle de Concorrência baseados em Timestamp

Diferente dos algoritmos baseados em bloqueios, os algoritmos de controle de concorrência baseados em *timestamp*, não estão interessados em manter a serialização por meio de exclusão mútua. Ao invés disso, eles selecionam uma ordem de serialização e executam as transações de acordo. Para estabelecer esta ordem, o gerenciador de transações, GT, associa a cada transação T_i um único *timestamp* fixo, denotado por $TS(T_i)$, em sua inicialização [ÖSZU 1999] [SILBERSCHATZ 1999].

Um *timestamp* permite identificar cada transação unicamente e sua ordenação. Há várias formas de associar *timestamps,* por exemplo, utilizar um contador global. Entretanto, a manutenção de contadores globais é um problema em sistemas distribuídos e móveis, sendo preferível que cada *site* tenha um contador local. Para manter a unicidade, característica importante deste mecanismo, cada *site* adiciona seu próprio identificador no contador. Assim um *timestamp* tem a forma <*valor do contador local, identificador do site*>. Formalmente, a regra de *timestamp* (TS) pode ser especificada como segue:

Dadas duas operações conflitantes O_{ij} e O_{kl} pertencentes, respectivamente, as transações T_i e T_k, O_{ij} é executada antes de O_{kl} se e somente se $ts(T_i) < ts(T_k)$. Neste caso T_i é dita ser a transação mais antiga e T_k é dita ser a transação mais jovem.

Um escalonador que segue o mecanismo de *timestamp* verifica para cada nova operação a existência de conflito com operações já escalonadas. Se a nova operação pertencer a uma transação que é mais nova do que todas as outras conflitantes, esta operação é aceita, caso contrário é rejeitada, fazendo com que ela seja reiniciada com um novo *timestamp*.

Um escalonador que opera desta forma garante escalonamentos serializáveis. Entretanto, comparações entre os *timestamps* das transações podem ser feitas somente se o escalonador já tiver recebido todas as operações a serem escalonadas. Se as operações chegarem no escalonador uma de cada vez, é necessário verificar se uma operação chegou fora da seqüência. Para facilitar esta verificação, a cada item de dados *x* são associados dois *timestamps*: um *timestamp* de leitura - *read timestamp rts(x)*, que é o maior dos *timestamp* das transações que leram *x*, e um de escrita – *write timestamp wts(x)*, que é o maior dos *timestamp* das transações que escreveram (atualizaram) *x*. Assim é suficiente comparar os *timestamps* de uma operação com os *timestamp* de escrita e leitura de um item de dados que quer acessar, determinando se alguma transação com um *timestamp* maior já acessou o mesmo item de dados.

4.2 Abordagem Otimista

Esta abordagem é utilizada quando a probabilidade de conflito entre transações é muito pequena, ao contrário do mecanismo de controle de concorrência pessimista, no qual os conflitos entre as transações são freqüentes.

A seqüência de operações da abordagem pessimista segue as seguintes fases: **validação** (V), **leitura** (L), **execução** (X) e **escrita** (E), como mostra a Figura 24.

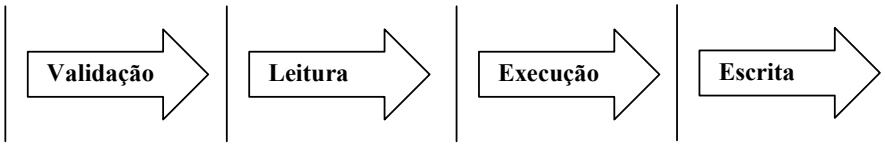

Figura 24 - Fases de execução pessimista de uma transação [ÖZSU 99].

Mecanismos otimistas, ao contrário, atrasam a fase de validação até antes da fase de escrita. Assim uma operação enviada para um escalonador otimista nunca é descartada. As operações de leitura, execução e escrita são processadas livremente sem atualizar o banco de dados atual. Cada transação faz as suas atualizações inicialmente nas suas cópias locais. A fase de validação consiste em verificar se estas atualizações não prejudicam a consistência do banco de dados. Se não prejudicam são então validadas globalmente. Caso contrário, a transação é cancelada e precisa reiniciar sua execução. A Figura 25 mostra as fases de execução otimista.

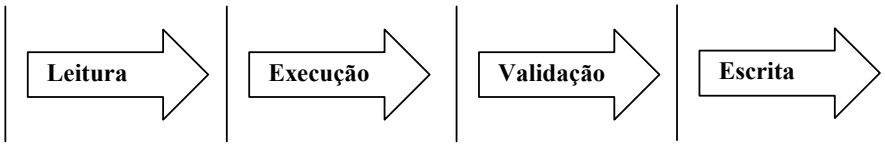

Figura 25 - Fases de execução otimista de uma transação [ÖZSU 99].

A idéia básica da abordagem otimista é que uma transação só é validada depois de já ter sido processada e antes da fase de sua atualização final (*commit*) no banco de dados. Desta forma se nenhum dos itens de dados foi modificado por outras transações, a transação pode ser consolidada. Em caso contrário ela será abortada. Devido a este modo de processamento, quando uma operação é enviada para um escalonador otimista ela nunca será descartada.

Nesta abordagem os mecanismos de controle de concorrência utilizados são os mecanismos de bloqueio e o de *timestamp.* O mecanismo de bloqueio utilizado é o já conhecido *bloqueio em duas fases*, mas com abordagem otimista, chamado de *Bloqueio otimista em duas fases para transações móveis*.

4.2.1 Mecanismos de Controle de Concorrência Otimistas baseados em Bloqueio

Os mecanismos de controle de concorrência otimistas baseados em bloqueio consistem em bloquear as unidades imediatamente, fazendo a verificação imediatamente antes da transação consolidar. O *algoritmo de bloqueio otimista em duas fases*, BO2F proposto por [CAREY & LIVNY, 1991] utiliza uma abordagem otimista de controle de concorrência "ler-um, escrever todos". No qual um bloqueio de leitura deve ser obtido imediatamente no *site* local da cópia, ou no *site* mais próximo, para cada operação de leitura. Os bloqueios de escrita para cópias replicadas são adiados até a fase de consolidação da transação. A utilização deste mecanismo torna a comunicação mais ágil, diminuindo a quantidade e custos de mensagens.

No entanto em um ambiente móvel a mobilidade das transações móveis deverá aumentar o envio de mensagens, aumentando assim o seu custo [JING et al., 1995]. Uma abordagem adequada para banco de dados, neste ambiente móvel, é o mecanismo de bloqueio otimista em duas fases para transações móveis, apresentado na seção a seguir.

4.2.2 Bloqueio Otimista em Duas Fases para Transações Móveis

O **BO2F-TM** [JING et al., 1995], *Bloqueio Otimista de Duas Fases para Transações Móveis,* possui uma grande vantagem perante os outros modelos, pois concede bloqueio de leitura imediatamente em demanda e, adia bloqueios de escrita até tempo de consolidação. Considerando um ambiente móvel onde as consultas aos itens de dados são muito mais freqüentes do que gravação de dados, este modelo requer menos mensagens que o mecanismo de controle de concorrência BO2F para banco de dados distribuídos [JING et al., 1995] [CAREY & LIVNY, 1991] .

O mecanismo utiliza cópias replicadas de itens de dados para reduzir os custos de mensagens impostas pela mobilidade das transações. O presente esquema de bloqueio, proposto por [JING 94], permite que uma operação de leitura desbloqueie um item de dados no *site* local (se contiver a réplica) e conceda a execução da operação de leitura. O desbloqueio pode ser feito em um *site* diferente do *site* no qual o bloqueio de leitura foi fixado.

O método de controle de concorrência otimista é geralmente satisfatório para aplicações de baixa contenção de dados. O BO2F-TM assume que bloqueios de leitura são executados imediatamente, quando são executadas operações em servidores de dados. Este fator é de extrema importância, pois em um ambiente móvel onde operações de leitura excedem em número as operações de escrita [JING et al., 1995] isto faz com que as unidades móveis tenham respostas muito mais rápidas às suas consultas. Além do mais, as transações buscam cópias consistentes de dados durante a sua execução.

O modelo BO2F-TM, emprega a abordagem de múltiplos coordenadores para as operações de uma transação, devido à mobilidade das unidades. Por exemplo, uma unidade móvel pode passar para uma célula nova, depois disto obter os resultados de operações previamente enviadas. Na célula nova, enviará o restante das operações de transação ao coordenador da atual MSS.

Este algoritmo emprega um método simples para restringir o número de mensagens extras de desbloqueio de leitura. Ao invés de enviar uma mensagem de desbloqueio de leitura ao servidor remoto da cópia, onde o bloqueio de leitura é feito, o BO2F-TM simplesmente **permite a execução da operação de desbloqueio** no *site* local ou mais próximo do servidor que contenha a cópia do item. Note o exemplo, demonstrado na Figura 26, como o algoritmo BO2F-TM desbloqueia na unidade atual os dados bloqueados em outros *sites*.

Considere um sistema de banco de dados móveis onde os itens de dados X e Y estão replicados nos *sites* A, B e C e o item de dados Z tem uma cópia no site C. A transação móvel T_1 requisita a operação de leitura r*ead*(X) no *site* A, se move para o *site* B e requisita a operação de leitura r*ead*(Y), no site B. Depois que a unidade se move para o *site* C, ela requisita a operação de escrita *write*(Z), seguida da operação de *commit*. Para poder ler cada item, o coordenador de cada unidade imediatamente requisita um bloqueio de leitura (*Read-Lock*) e uma operação de leitura no servidor local. Quando for consolidar a transação, no mecanismo de B2F-D, o coordenador precisa enviar

mensagens de *commit* (ou desbloqueio) para os servidores A e B para a liberação dos bloqueios de leitura.

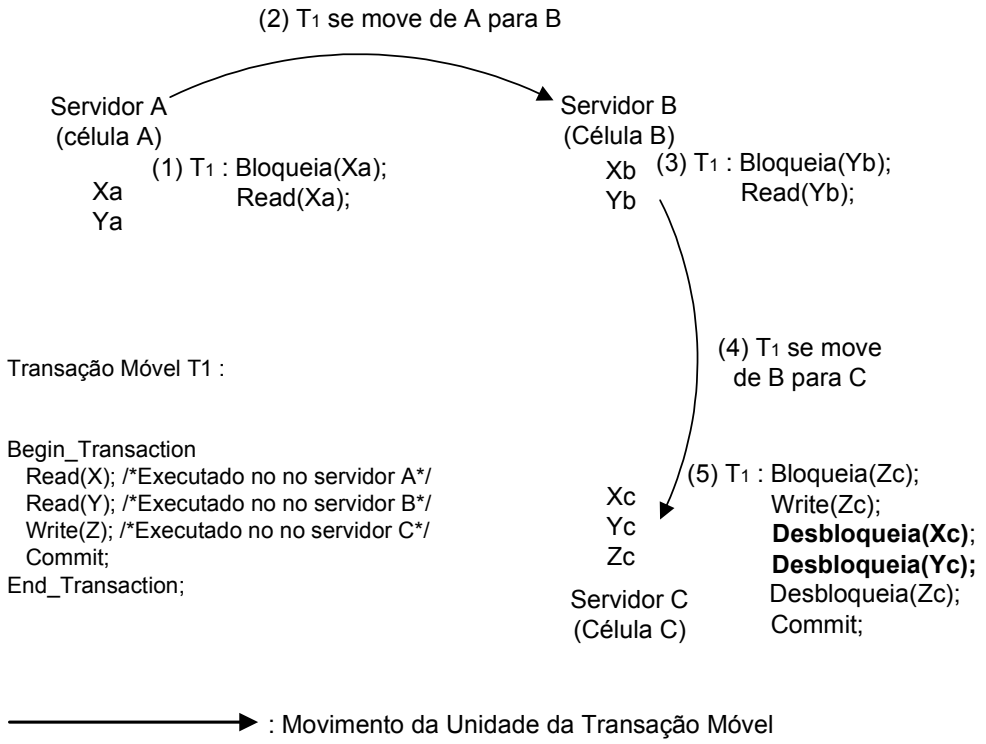

Figura 26 - TM utilizando o BO2F-TM [CAREY & LIVNY, 1991].

Com a utilização do algoritmo BO2F-TM ao invés de requisitar o desbloqueio de (Xa) e desbloqueio de (Yb) nos servidores A e B, como é feito no mecanismo de B2F-D, o BO2F-TM executará desbloqueio (Xc) e o desbloqueio (Yc) no servidor local C, a tempo consolidar, sem a necessidade de transmitir mensagens. E executa a operação de escrita, *write*(Zc), seguido de uma operação de *commit*, consolidando assim a transação.

Este exemplo mostra que, com a introdução de transações móveis, o BO2F-TM emprega um método simples para restringir o número de mensagens extras de desbloqueios de leitura.

4.2.2 Ordenação por Timestamp para Transações Móveis

No algoritmo de *timestamp otimista para transações móveis* TO/TM de [LEE et al., 2000], o gerenciador de transações na MSS associa um único *timestamp* - ts(TM_i) - para cada transação móvel. Os *timestamps* são gerados em ordem crescente. Com o objetivo de solucionar os conflitos de acesso aos dados, o TO/TM mantém para cada objeto de dados *x* um número máximo de *timestamps* para operações de leitura e escrita, denotados por *max-read-ts*[x] e *max-write-ts*[x], respectivamente. Os índices *max-read-ts*[x] e *max-write-ts*[x] designam os *timestamps* mais recentes para leituras e escritas que acessam o objeto de dado x.

Um escalonador de transações em TO/TM escalona as operações emitidas das transações móveis sem nenhum atraso, ou seja, quando o escalonador recebe uma operação do gerenciador de transações, ele não decide se aceita ou não esta operação, mas a escalona imediatamente. No esquema de *timestamp* tradicional, receber uma operação, o escalonador deverá determinar se aceitará ou rejeitará a mesma.

Quando uma transação móvel, TM_i, está pronta para entrar na fase de consolidação, o escalonador verifica se os valores de *max-read-ts* e *max-write-ts* dos objetos que foram lidos e escritos por TM_i foram alterados por uma outra transação enquanto estavam sendo atualizados por TM_i, e ,em caso afirmativo, o escalonador rejeita a operação de *commit* de TM_i e aborta TM_i. Caso contrário o escalonador consolida a TM_i passando a operação de *commit* para o gerenciador de dados, permitindo assim que a TM_i termine com sucesso.

Com a finalidade de verificar se a TM_i possa ser seguramente consolidada ou se ela deva ser rejeitada, o TO/TM emprega várias estruturas de dados para cada transação ativa, TM_i, o que inclui não somente os objetos de dados lidos e escritos por TM_i , mas também os valores de *max-read-ts* e *max-write-ts* dos objetos de dados correspondentes.:

- Uma tabela contendo os nomes e os *timestamps* das transações ativas;
- Um conjunto consistindo das operações de leituras que foram escalonadas de TM_i, denotado por *read-scheduled*[TM_i];

- E outro conjunto das escritas escalonadas de TM_i , denotado por *write-scheduled*[TM_i] .

Quando o escalonador recebe uma operação de leitura (r_i[x]) ou de escrita (w*i*[x]) da TM_i, ele adiciona *x* e o valor de *max-read-ts*[x], ou *max-write-ts*[x], ao grupo de leituras escalonadas*, read-scheduled*[TM_i] (ou *write-scheduled*[TM_i]). Já quando o escalonador recebe uma operação de *commit*$_i$, isto é, MT_i terminou sua execução, então as leituras e escritas de TM_i , escalonadas, são adicionadas aos conjuntos de *read-scheduled*[TM_i] e *write-scheduled*[TM_i] respectivamente.

Para resolver conflitos de escrita-leitura o valor do *max-write-ts* para cada objeto de dados *x* no grupo de leituras escalonadas, *read-scheduled*[TM_i], é comparado com o valor atual do *max-write-ts* do objeto. Se o valor do *max-write-ts*[x] no grupo de leituras escalonadas de [TM_i], é idêntico ao valor corrente do seu *max-write-ts*, a operação de leitura do objeto de dado pode ser considerada ter sido corretamente executada. Caso contrário, a TM_i deverá ser abortada, pois a operação foi incorretamente executada.

Com o objetivo de verificar os conflitos de leitura-escrita e escrita- escrita, os valores armazenados de *max-read-ts* e *max-write-ts* para cada objeto *x* no grupo de escritas escalonadas, *write-scheduled*[TM_i] são comparados com os valores correntes de ambos *max-read-ts* e *max-write-ts*, respectivamente. Se os valores dos *max-read-ts*[x] e *max-write-ts*[x] no grupo de escritas escalonadas, *write-scheduled*[TM_i], são equivalentes aos valores correntes dos seus *max-read-ts* e *max-write-ts*, a operação de escrita no objeto de dados foi corretamente executada. Caso contrário a TM_i deverá ser abortada pelo escalonador.

Em [LEE et al., 2000] é apresentado o algoritmo de controle de concorrência do mecanismo de TO/TM, que é executado no servidor de cada estação base. Este mecanismo consiste de dois componentes distintos: UMs e MSSs. Cada unidade móvel é composta de três componentes: G*erador de Transações Locais (*GTL), *Coordenador de Transações Locais* (CTL) e um *Gerenciador de Cache* (GC).

Cada MSS possui três componentes: *Gerente de Transações Móveis* (GTM), *Gerenciador de Replicas* (GR) e o *Gerenciador de Dados* (GD). O GTL é responsável pela geração de transações locais, que são modeladas com seqüências de operações de *read* e *write* . O GTL gerencia as transações locais do início até suas consolidações.

O GTM tem a responsabilidade de escalonar as transações validando suas execuções de acordo com a abordagem em questão. O GR mantém consistência mútua de objetos replicados armazenados na MSS e o GD é responsável pelo processamento dos dados nos servidores de discos.

4.2.3 Controle de Concorrência na arquitetura Broadcast

Em contraste com a tradicional arquitetura cliente/servidor, onde os dados são entregues do servidor para os clientes por demanda (o cliente requisita dados para o servidor explicitamente), uma grande quantidade de aplicações clientes beneficia-se de uma modalidade diferente obtenção de dados: a disseminação.

Nestas aplicações, o servidor transmite dados a uma população de clientes, sem que estes tenham feito uma requisição específica. Os clientes monitoram o canal de *broadcast* e recuperam os itens de dados que eles necessitam logo que estes se tornam disponíveis. Tais aplicações envolvem tipicamente um número pequeno de servidores e um número muito maior de clientes com interesses similares.

Com a utilização de broadcast dos dados, as transações móveis não necessitam informar o servidor de broadcast antes de acessar um item de dados. Porém enquanto os itens de dados estão sendo transmitidos através de *broadcast*, transações de atualizações no servidor de banco de dados podem estar alterando os valores destes itens de dados

Para manter a validade e consistência dos itens de dados em um banco de dados em constante atualização (informação meteorológicas, mercado de ações, sistemas de controle de tráfego, etc.), transações de atualização são geradas para atualizar os valores dos dados sempre que os estados destes objetos sofrem modificações no ambiente

externo. Cada transação de atualização tem uma marca de tempo. A marca de tempo é utilizada para indicar em qual momento a atualização é gerada. No item de dados atualizado é registrado um novo valor no seu número de versão.

O servidor de banco de dados executa periodicamente o *broadcast* dos itens do banco de dados. A cada período de *broadcast* dá-se o nome de **ciclo de *broadcast***. O comprimento de um ciclo de *broadcast* pode ter duração fixa ou variar de acordo com o tamanho dos itens de dados enviados. O processo de *broadcast* é modelado como uma transação de *broadcast*. O conjunto de itens de dados na transação de *broadcast* é formado pelos itens de dados disseminados num ciclo. A transação de broadcast é executada de forma entrelaçada com a transação de atualização. Com isso é reduz o atraso dos bloqueios nas transações de atualização.

Neste modelo é assumido que as transações geradas nos clientes móveis são de *somente leitura* e as transações de atualizações são curtas, consistindo de uma seqüência de operações de *read* e *write*. Além disso, é assumido que o protocolo de controle de concorrência de bloqueio em duas fases no método otimista é utilizado para o controle de concorrência entre as transações de atualizações, que por sua vez são realizadas somente nos servidores de bancos de dados [LAM et al., 1999a], [LAM et al., 1999b] e [PITOURA & CHRYSANTHIS, 1999b].

As suposições do modelo de transação são sumarizadas a seguir:

- Todas as transações de atualização são executadas no servidor de banco de dados;
- Todas as transações móveis são somente-leitura e seus conjuntos de leituras são *desordenados*, ou seja, itens de dados requisitados pela unidade móvel podem ser recebidos em qualquer ordem;
- Os conflitos entre transações de atualizações e transações móveis são baixos, que é o caso da maioria dos sistemas de *broadcast* tais como sistemas de informação sobre tráfego e sobre o tempo;

- Transações móveis possuem um período limite para completar sua execução, um *deadline*. É importante completar a transação antes de seu *deadline*, caso contrário ela deverá ser reiniciada.

Para o controle de concorrência entre as transações móveis e as transações de atualização, no ambiente móvel com utilização de *broadcast* são propostos os seguintes mecanismos: *Update First with Order* (UFO) [LAM et al., 1999a], *Ordered Update First with Order* (OUFO) [LAM et al., 1999b]

4.2.4 UFO - Update First Order

O maior problema no projeto de protocolo de controle de concorrência para sistemas de computação móvel, utilizando *broadcast* de dados, está no servidor de banco de dados que geralmente não percebe que itens de dados estão sendo lidos por uma transação móvel e em que momento. Isto faz com que seja difícil detectar itens de dados conflitantes. No ambiente móvel a largura de banda dos canais móveis, especialmente para *up-link*[4], é muito limitada. Por isso, é importante que se minimize o custo de comunicação entre os clientes móveis e o servidor de *broadcast*, especialmente usando canais de *up-link*. O mecanismo UFO checa conflitos de dados entre transações de *broadcast* e de atualizações ao invés de detectar conflitos entre transações móveis e de atualização [LAM et al., 1999a].

O princípio básico deste mecanismo é assegurar que se um conflito de dados ocorre entre uma TB (transação de *broadcast*) e uma TA (transação de atualização), a serialização imponha a seguinte ordem: TA $\rightarrow$ TB. O caso de TB $\rightarrow$ TA só será permitido se for impossível uma transação móvel ler dados de uma TB, que sejam diretamente dependentes de uma TA.

Para o caso de conflito entre uma transação móvel (leitura) e uma transação de *broadcast*, a ordem da serialização entre TB e TM sempre será TB $\rightarrow$ TM. Assim, em

[4] Up-link – ligação do cliente móvel para o servidor.

caso de conflito, a ordem de execução entre transações de atualização e transações móveis sempre será TA → TM, assegurando a serialização.

Basicamente o algoritmo UFO consiste em duas partes:

- Execução da transação de atualização;
- Resolução do conflito entre a transação de atualização e a transação de broadcast.

4.2.4.1 Execução das Transações de Atualização

A execução de uma transação de atualização é dividida em duas fases: a *fase de execução* e a *fase de atualização*. Existem duas vantagens importantes em dividir a execução de uma transação de atualização em duas fases. Primeiro reduz significativamente a probabilidade de bloqueio das transações de *broadcast*. Segundo, a detecção dos conflitos de dados entre uma transação de atualização e uma transação de *broadcast*, se tornará mais fácil.

Durante a fase de execução, as operações de uma transação de atualização são executadas e qualquer dado que está em conflito com outras transações de atualizações são resolvidos usando o protocolo de controle de concorrência de bloqueio em duas fases. As atualizações permanentes no banco de dados são feitas durante a fase de atualização.

As atualizações dos itens de dados são escritas primeiramente em um *workspace* (espaço de trabalho) privado, ao invés de atualizar imediatamente o banco de dados. Quando todas as operações de atualização são completadas, os dados que estão em conflito com a transação de *broadcast* são conferidos. Atualizações se tornam permanentes no banco de dados copiando os valores do espaço de trabalho privado para o banco de dados. Para solucionar conflitos de dados com outras transações de atualizações esse mecanismo utiliza bloqueio em duas fases na abordagem otimista.

4.2.4.2 Apresentação do Algoritmo UFO - Update-First with order

Antes do início da fase de atualização, o sistema detecta os conflitos de dados entre as transações de atualização e as transações de *broadcast*. Se uma operação de *broadcast* quer ler um item de dado que está bloqueado por uma transação de atualização, a transação de *broadcast* será bloqueada até que a transação de atualização seja consolidada, seguindo o princípio do bloqueio em duas fases.

Para detectar conflitos entre transações de atualização e transações de *broadcast*, o sistema compara o conjunto de escrita de uma transação de atualização com o conjunto de leitura de uma transação de *broadcast*. O sistema pode determinar se há algum dado em conflito entre os conjuntos.

Conflitos são resolvidos na fase de atualização. A idéia básica é reverter a ordem de serialização de TB $\rightarrow$ TA na ordem desejável TA $\rightarrow$ TB. O seguinte algoritmo definido por [LAM et al., 1999a] é executado no servidor de *broadcast*. Antes de sua apresentação são definidos alguns elementos:

TB = $\sum_{J=0}^{nc} TB_{i-j}$

para qualquer ciclo corrente **i** de broadcast

$\mathbf{O_{TB}}$ = conjunto de itens de dados de uma transação de *broadcast*, TB

$\mathbf{O_{TA}}$ = conjunto de itens de dados de uma transação de atualização, TA

$\mathbf{B_A}$ = $\{x \in O_{TB} \cap O_{TA} \mid x$ já foi para broadcast quando TA chega$\}$

Se $O_{TB} \cap O_{TA} = \{\}$
    Então TB e TA não tem dependência
Senão
    Se $B_A = \varnothing$
        Então a ordem de serialização é TA $\rightarrow$ TB
    Senão
        Para cada item de dado i $\in B_A$
            Re-broadcast o item de dado i
        Até a ordem de serialização for TA $\rightarrow$ TB
    Fim Se
Fim Se

Segundo [LAM et al., 1999a] as principais considerações sobre o algoritmo UFO são:

- O impacto no algoritmo de *broadcast* adotado é mínimo;
- É assegurado que todos os escalonamentos entre as transações móveis e transações de atualizações serão serializáveis.
- A implementação do algoritmo UFO é simples e não requer nenhuma alteração por parte dos clientes móveis.

4.2.5 Ordered Update First Order - OUFO

Em [LAM et al., 1999b] o algoritmo *Ordered Update First Order* (OUFO), uma extensão do modelo UFO, é proposto para fazer o broadcast de forma consistente dos itens de dados enquanto atualizações são feitas concorrentemente no servidor de *broadcast*. Ao contrário do algoritmo UFO as operações *read* são executadas de forma ordenada.

Além disso, neste algoritmo é desenvolvido um esquema de relatório de invalidação para assegurar a consistência de itens de dados armazenados localmente no cliente móvel e também busca reduzir o *overhead* de reinícios de transações efetuando o *rebroadcast* somente no item de dados conflitante.

Algumas considerações sobre este mecanismo:

- É proposto para as transações móveis cujas operações de leitura de uma transação são *ordenadas*.
- Um esquema híbrido *rebroadcast/invalidação* é desenvolvido para assegurar a consistência dos dados armazenados nos clientes móveis e busca minimizar o custo de reinícios de transações devido aos conflitos de dados;
- Todas as transações de atualização são curtas e são processadas no servidor de broadcast;
- A taxa de chegada das transações de atualização é alta e esporádica;
- Todas as transações móveis possuem um tempo limite de execução, o *deadline*. É importante que uma transação termine sua execução antes do seu *deadline*. Caso contrário esta transação precisa ser reinicializada;

- O resultado de uma transação móvel é reportado para o cliente requisitante em tempo de consolidação da transação;
- Cada cliente móvel mantém somente aqueles dados acessados recentemente em *cache*.
- As requisições feitas pelo processamento de uma transação móvel asseguram que todos os itens de dados acessados são consistentes e ao mesmo tempo a concorrência dos dados é maximizada;
- O processo de *broadcast* é modelado como uma *transação de broadcast* (TB) ;

O princípio básico utilizado no algoritmo OUFO para assegurar a consistência dos dados é que, na ocorrência de um conflito entre uma transação de *broadcast* e uma transação de atualização, a ordem de serialização entre elas deve ser TA $\rightarrow$ TB. Considerando que transações móveis lêem itens de dados apartir de transações de *broadcast*, a ordem de serialização entre TA e transação móvel, TM, deve ser sempre TA $\rightarrow$ TM. Assim os escalonamentos serão sempre serializáveis. Basicamente o método UFO consiste em três partes:

- Execução das transações de atualização;
- Resolução de conflitos entre as transações de atualizações e transações de *broadcast;*
- A consistência é verificada para os itens armazenados localmente nas estações móveis;

Execução das Transações de Atualização - A execução das transações de atualização no algoritmo OUFO é feita da mesma forma como no algoritmo original. É dividida em duas fases (execução e atualização). Durante a fase de execução, as operações de uma transação de atualização são executadas e os conflitos de dados com outra transação de atualização são solucionados usando um mecanismo convencional de controle de concorrência como o mecanismo de bloqueio em duas fases.

Resolução de Conflitos entre Transações de Atualização e Transações de Broadcast - Os conflitos entre uma transação de atualização e uma transação de broadcast serão

detectados quando a transação de atualização entrar na sua fase de atualização em que um esquema de rebroadcast é utilizado para resolver conflitos. O propósito é reverter a ordem de serialização de TB $\rightarrow$ TA para a ordem desejada, TA $\rightarrow$ TB. Os detalhes do algoritmo de ambos servidor de broadcast e clientes móveis são apresentados abaixo.

Verificação da Consistência de itens armazenados localmente nos cliente móveis - Se uma transação móvel precisa ler todos os seus itens de dados requisitados de uma transação de broadcast, a transação móvel poderá ter que esperar por um longo período para conseguir acesso aos seus dados requeridos. Este tempo de espera depende do algoritmo de broadcast adotado e obviamente do tamanho do banco de dados. Um outro problema de performance do algoritmo OUFO é que uma transação móvel precisa ser reiniciada se qualquer dos seus itens de dados acessados for submetido a uma transação de broadcast de novo antes de a transação móvel consolidar. Reinicializando uma transação móvel pode aumentar muito o seu tempo de resposta, uma vez que as transações móveis precisam esperar pelos seus itens de dados requeridos durante diversos ciclos de broadcast. No pior caso, ela pode ser re-escalonada repedidas vezes até perder o seu tempo limite, o seu deadline.

Com o objetivo de reduzir este delay de acesso aos dados e reduzir o custo de reinícios, um cliente móvel deverá manter os dados acessados recentemente em sua cache. No caso de uma transação móvel ser reiniciada devido a chegada de uma nova versão de um de seus itens de dados acessados, esta transação pode imediatamente acessar este item de dados de sua cache (se ele já tiver sido recentemente utilizado) ao invés de ficar esperando uma transação broadcast que atualize o valor deste dado. O algoritmo OUFO utiliza um mecanismo de invalidação (descrito em detalhes em [LAM et al., 1999a]) que foi desenvolvido para assegurar que todos os itens de dados que estão na cache são consistentes. Além disso, depois que uma transação móvel acessa um item de dado, este dado também é colocado na cache do cliente móvel. Quando a cache está cheia, um algoritmo de substituição, como o método do *Least Recently Used* (último item de dado acessado recentemente), é chamado.

Neste mecanismo dois algoritmos são executados. Um deles no Servidor de Broadcast e outro nos clientes móveis. Esses algoritmos são apresentados a seguir.

4.2.5.1 Algoritmo no Servidor de BroadCast

É executado quando a transação de atualização entra na fase de atualização [LAM et al., 1999b].

O_{TB} = conjunto de itens de dados da transação de broadcast, TB
O_{TA} = conjunto de itens de dados da transação de atualização, TA

Se $O_{TB} \cap O_{TA} = \varnothing$
    Então TB e TA não tem dependência
Senão
    Para cada item de dado $d_i \in \{ O_{TB} \cap O_{TA} \}$
        Re-broadcast o item de dado d_i
Fim Se

Note que na fase de atualização, o conjunto de itens de dados a serem atualizados por uma transação de atualização já está definido. Através do *rebroadcast* os itens de dados conflitantes, a ordem de serialização entre as transações de *broadcast* e as transações de atualização são invertidas.

4.2.5.2 Algoritmo no Cliente Móvel

Os itens de dados requisitados por uma unidade móvel são representados por uma seqüência de operações de leitura. O processamento de uma TM começa da primeira operação de leitura até a última, em seqüência. Cada item de dados recebido de uma TB é comparado com o item de dados requisitado da operação de execução. Se eles combinam, a TM irá ler os itens de dados e as operações serão processadas. O processo de comparação é repetido para a próxima operação de leitura até o final de seqüência, e então a transação é consolidada.

No caso de um item de dados que foi lido por uma TM, for *rebroadcast* enquanto a TM está esperando por outro item de dados, a TM será reiniciada pelas operações que requisitaram aquele item. Ela irá utilizar o valor de *rebroadcast* para a execução da operação. Há duas razões para o reinicio da transação: é necessário assegurar ordem de serializabilidade desejada TA $\rightarrow$ TM e fornecer a maioria dos valores de dados atualizados para as transações móveis.

No algoritmo apresentado a seguir, também porposto em [LAM et al., 1999b], assume-se que uma TM lê todos os itens de dados de uma TB e não existe armazenamento pelo lado cliente.

L_{TM} = a seqüência de operações de leitura na TM
S_{TM} = o conjunto de itens de dados que foram acessados pela TM
d_c = o item de dados requerido pela primeira operação em LTM
S_{TM} = {}

---

repita L_{TM}
leia item de dados d_i da TB (d_i é o item de dados difundido correntemente)
se $d_i = d_c$
então TM processa d_i e atualiza L_{TM}
senão
se $d_i \in S_{TM}$
TM repete o processamento de d_i e atualiza L_{TM}
Reinicia sua execução apartir da operação que requisita d_i
fim se
fim se
d_c = o item de dados requisitado pela próxima operação em L_{TM}
fim repita

Com o objetivo de manter a ordem de execução das operações, operações dependentes de uma operação abortada também são abortadas. Uma transação móvel é reiniciada tantas vezes quanto seus itens de dados acessados forem atualizados antes da terminação. Entretanto, este é o custo pago por assegurar a correção dos resultados e as propriedades ACID. Para minimizar os impactos e custos o OUFO mantém dados armazenados localmente, assim uma transação reiniciada somente precisa de um curto tempo para retornar ao seu ponto de reinício.

4.2.6 Método de Checagem de Serialização - MCS

O problema de performance encontrado do algoritmo UFO é causado pela sensibilidade à probabilidade de conflitos de dados entre uma transação de atualização e transação de *broadcast* (grande quantidade de conflitos implica em aumento do número de mensagens e degradação da performance do sistema). No MSC proposto em [AU et al.,

2001], este problema é amenizado porque a consistência dos valores de dados observados para uma transação móvel é assegurado, executando uma checagem no gráfico local de serialização, em vez de confiar somente nos dados modificados por transações de *broadcast*.

O principal problema da inconsistência de dados no ambiente de *broadcast*, é que itens de dados capturados no "meio" de um processo de *broadcast*, podem ser utilizados depois por uma transação de atualização. Se as transações de atualização tiverem dados adicionais e estes estiverem em conflito com a transação de *broadcast*, pode ocorrer um escalonamento cíclico com a TB. Outra razão para o escalonamento cíclico é a dependência transitiva sobre a transação de atualização. Estes dois problemas podem proporcionar para o cliente móvel, conflito de dados de informação das transações de atualizações, para a construção de seus gráficos locais de serialização. No caso de um escalonamento cíclico, o cliente móvel irá dispor de um item de dado de leitura antes da transação de atualização. Isto causa um escalonamento não serializável.

Para resolver o primeiro caso, o MCS faz um *rebroadcast* das identidades dos itens de dados que são atualizados por uma transação no servidor de banco de dados. Para controlar o caso das dependências transitivas, o MCS simplesmente faz um *broadcast* das transações de atualizações que foram difundidas no último período.

4.2.6.1 Algoritmo para o Servidor

Da mesma forma que o algoritmo UFO, o MCS divide a execução das transações de atualizações em duas fases. Quando houver um conflito de entre uma transação de atualização e uma transação de *broadcast*, este será descoberto quando a transação de atualização entrar na fase de atualização. No caso de dados em conflito, a dependência entre a transação de atualização e a transação de *broadcast* será dividida em vez de executar o *rebroadcast* dos itens de dados em conflito.

Depois da conclusão da transação de atualização o seguinte algoritmo é executado. Neste algoritmo, proposto em [AU et al., 2001], assume-se que:

$\mathbf{D_{TA}}$ = conjunto de itens de dados da transação de atualização, TA
$\mathbf{D_{TB}}$ = conjunto de itens de dados da transação de atualização no último período depois da queda
$\mathbf{T_B}$ = conjunto das transações de atualização do último período

---

se $D_{TB} \cap D_{TA} \neq \varnothing$
  então faça *broadcast* da mensagem: D_{TA} está atualizado
senão **se** $D_{TA} \cap D_{TAi} \neq \varnothing$ onde $D_i \in T_B$
  então faça *broadcast* da mensagem: D_{TA} está atualizado
  fim se
fim se

Transações movéis executam transações de broadcast para seus itens de dados requeridos e também executam transações de atualização para construir gráficos de serialização local. Atualizado o gráfico de serialização, checa-se a existência de escalas cíclicas. Em caso afirmativo, a transação móvel resolve isto, dispondo dos itens capturados antes da transação de atualização, o que causa um escalonamento não serializável. Os itens de dados dispostos serão capturados do "meio" quando houver a próxima difusão. O processo continuará até que todos os dados exigidos sejam capturados ou o período de *deadline* da transação for expirado.

4.2.7 Mecanismo de Broadcast Multiversão

Com o objetivo de minimizar abortos de transações móveis, versões de itens de dados mais antigas são retidas temporariamente. O servidor de broadcast, ao invés de fazer o processo de *broadcast* do valor mais atual para cada item de dados, mantém e difunde versões múltiplas de cada item, chamado *Mecanismo de Broadcast Multiversão* proposto em [PITOURA & CHRYSANTHIS, 2002]. Esquemas multiversão são úteis principalmente para propósitos de aumentar a concorrência, aumentando assim a performance e tempo de resposta.

Em um ciclo de *broadcast* v_0 uma transação móvel somente leitura *TR* executa sua primeira operação de leitura. Durante v_0, *TR* lê as versões de dados mais recentes, ou seja, as versões com o número de versão maior. Nos subseqüentes ciclos de *broadcast*, para cada item de dados lidos por *TR*, contidos no conjunto de leitura de *TR*, esta deverá

ler a versão com o número de versão maior, v_c, sendo que esta versão deve ser maior ou igual a v_0. Se tal versão existe no canal de *broadcast, TR* continua, senão *TR* é abortada.

4.3 Abordagem Híbrida

Métodos de controle de concorrência otimistas estão ganhando popularidade, especialmente com banco de dados distribuídos e em ambientes móveis, que confiam em técnicas otimistas para aumentar a disponibilidade e a performance do sistema de banco de dados.

A abordagem otimista geralmente é utilizada quando a unidade móvel está desconectada da rede e precisa fazer o processamento das transações com os dados armazenados localmente, contando com alta contenção de dados e conseqüentemente poucos conflitos. O maior problema com a técnica otimista é que ela não fornece resultados satisfatórios em ambientes com grande ocorrência de conflitos, ao contrário da técnica pessimista, especialmente com bloqueios, que se desenvolve muito melhor nestas circunstâncias.

A técnica híbrida explorada por [PHATAK et al., 1995] fornece bloqueios para itens de dados em ambientes altamente conflitantes na técnica pessimista, e acesso otimista para o restante, conseguindo assim um melhor desempenho do sistema [GUPTA et al., 1997]. A idéia básica é que uma transação seja pessimista enquanto a unidade móvel está executando em modo conectado, e otimista quando a unidade móvel estiver em modo desconectado. De acordo com [PHATAK et al., 1995], o projeto do *gerenciador de bloqueios* é a chave para esta técnica híbrida. O gerenciador de bloqueio mantém um buffer finito de bloqueio. Cada *slot* (ou frame) no *buffer* de bloqueio detém bloqueios e requisições de bloqueios pendentes para um único item de dados. O número de itens de dados com requisições de bloqueios ativos não pode exceder ao número de *slots* (espaços) no *buffer*. Assim, o número de *slots* no *buffer* é menor que o número de itens de dados, mas é maior que o número esperado de conflitos de itens de dados.

Sempre que uma requisição de bloqueio para um item de dados *x* for recebida, o gerenciador de bloqueios primeiro tenta localizar *x* no *buffer* de bloqueio. Se for encontrado o gerenciador de bloqueios tenta salvar o bloqueio no *slot* correspondente.

A requisição de bloqueio pode ser concedida, bloqueada ou colocada como bloqueio pendente. Assim para a operação de *read* (leitura compartilhada) é concedido um bloqueio, se não houver nenhum bloqueio exclusivo existente em *x*. Um bloqueio de operação de *write* (escrita exclusiva) é concedido, se não houver nenhum bloqueio existente em *x*. Se *x* não estiver no *buffer* de bloqueio, um *slot* deve ser alocado para salvar esta requisição de bloqueio. Se existir um *slot* livre este será usado, caso contrário um *slot* vítima deve ser selecionado.

Qualquer *slot* sem bloqueio ou requisição de bloqueio é considerado livre. O item de dados no *slot* vítima selecionado é rejeitado. Todos os bloqueios relativos a ele são removidos e todas as transações bloqueadas são liberadas. Sempre que uma requisição de bloqueio for rejeitada, a transação que originalmente requisitou o bloqueio utiliza a abordagem otimista com relação ao item de dados. O bloqueio para *x* é posto no *slot* vítima. Se a utilização do *buffer* de bloqueio é zero, então todas as requisições de bloqueio são rejeitadas e não existirá nenhum bloqueio ativo. Neste cenário todas as transações utilizam a abordagem otimista com relação aos itens de dados de seu conjunto de leitura e escrita e o sistema fica puramente otimista. Porém, se algum item de dados precisar ser despejado do buffer de bloqueio (um bloqueio nunca seja rejeitado) o sistema passa a atuar de forma pessimista.

Ao ser consolidada, uma transação deve ser validada com relação a todos os itens de dados dos conjuntos de leitura e escrita. Considerando uma transação T e uma transação concorrente T', que iniciou sua execução antes de T ser consolidada, a transação T é válida com relação a um item de dados *x*, se uma das condições a seguir for satisfeita:

1 – T detém um bloqueio em x.
2 – *x* não está no conjunto de escrita de T. T não detém o bloqueio sobre x e as condições são atendidas:

(a) se outra transação T' detém um bloqueio em x, então não é um bloqueio exclusivo;

(b) *x* não foi sobrescrito depois que foi lido por T (não está no conjunto de escrita de qualquer transação concorrente e consolidada).

3 – *x* está no conjunto de escrita de T, T não detém um bloqueio em x e as seguintes condições são atendidas:

(a) nenhuma outra transação detém um bloqueio em *x*;

(b) *x* não foi sobrescrito depois que foi lido por T.

Depois que a validação obtiver sucesso, o conjunto de escrita é gravado no banco de dados e todos os bloqueios são liberados. As condições 2 e 3, de que T não detenha um bloqueio sobre x, são desnecessárias, desde que a condição 1 impeça qualquer subcondição das condições 2 e 3. Estas condições estão incorporadas no algoritmo de validação [PHATAK et al., 1995], no qual:

- *READSET(T)* é o conjunto de itens de dados lidos por T;
- *WRITESET (T)* é o conjunto de dados de escrita por T;
- *LOCKSET(T)* é o conjunto de todos os itens que T possui bloqueios; e
- WLOCKSET(T) $\subseteq$ LOCKSET(T) é o conjunto de todos os itens de dados que T possui bloqueios exclusivos na fase de validação.

```
//Entrada: Transação T
para todos x ∈ READSET (T) faça
    se x ∉ ao LOCKSET (T) então
        se x ∉ WRITESET (T) e ∃ T′: x ∈ WLOCKSET(T′) então
            aborte T
            retorne
        fim se
        se x ∈ WRITESET (T) e ∃ T′: x ∈ WLOCKSET(T′) então
            aborte T
            retorne
        fim se
        se ∃ consolidado T″ concorrente de T: x ∈ WRITESET (T″) então
            aborte T
            retorne
        fim se
    fim se
fim para
Commit T
```

Fim

Note que ambos os conjuntos de bloqueio (*LOCKSETS*) não contêm itens de dados nos quais seus bloqueios foram rejeitados. E também para este modelo WRITESET(T) $\subseteq$ READSET(T).

Uma transação T' concorrente de T, é qualquer transação cujo tempo de vida se sobreponha a ao de T. Qualquer transação que possua bloqueios quando T é validada, é automaticamente concorrente de T (desde seja ativa para obter bloqueios). Este mecanismo prioriza acesso pessimista sobre acesso otimista aos dados. A condições de validação previnem qualquer transação otimista com relação a um dado x, utilizando-o para escrita, quando outra transação detiver um bloqueio em x (compartilhado ou exclusivo).

Similarmente, se uma transação possui um bloqueio exclusivo (*write*) em x, então nenhuma outra transação que tenha lido *x* pode ser consolidada até que este bloqueio seja liberado. Conflitos puramente otimistas são tratados usando as condições 2b e 3b, que são as únicas relevantes quando o *buffer* de bloqueio é de tamanho zero e o sistema é puramente otimista.

Um detalhe fundamental do algoritmo é que, se bloqueios e requisições de bloqueios podem ser rejeitados, a transação pode somente requerer um bloqueio uma vez no item de dados. Caso contrário o escalonamento não é serializável. O pedido de atualização é tratado como uma requisição de bloqueio, espera-se que no caso do item de dados não estiver no buffer de bloqueio, a requisição é "rejeitada" sem selecionar um *slot* vítima, e a transação tem que continuar sendo otimista com respeito a este item de dados. Uma simulação do diagrama de blocos é dada na Figura 27. Os blocos básicos deste modelo de simulação são descritos a seguir:

1. **Fonte:** Cria todas as transações no sistema, seguidas das listas de tuplas (itens de dados) que estas transações irão acessar.

2. **Gerenciador de Transações:** Este módulo fornece todas as funcionalidades de um gerenciador de transações para cada transação, incluindo habilitação para requisições de bloqueios, requisições de páginas, gerenciamento de *deadlocks* e as fases de leitura/escrita e validação das transações. Estas funções são executadas pela CPU.
3. **CPU:** Cada CPU tem a capacidade de multitarefa. E executa funções fornecidas pelo gerenciador de transações.
4. **Controle de Concorrência:** Este módulo gerencia o *Buffer de Bloqueios*, obtém os bloqueios e requisita os desbloqueios, e desenvolve as fases de validação das transações.
5. **Gerenciador de Buffer:** Gerencia uma *cache.* Cada *slot* na *cache* detém uma página. Todas as requisições de E/S (entrada/saída ou I/O) das transações vão direto para a *cache do buffer*.
6. **Gerenciador de disco:** Gerencia os acessos a disco. As páginas da relação estão espalhadas pelos discos. Somente o gerenciador de *buffer* pode acessar os discos diretamente.

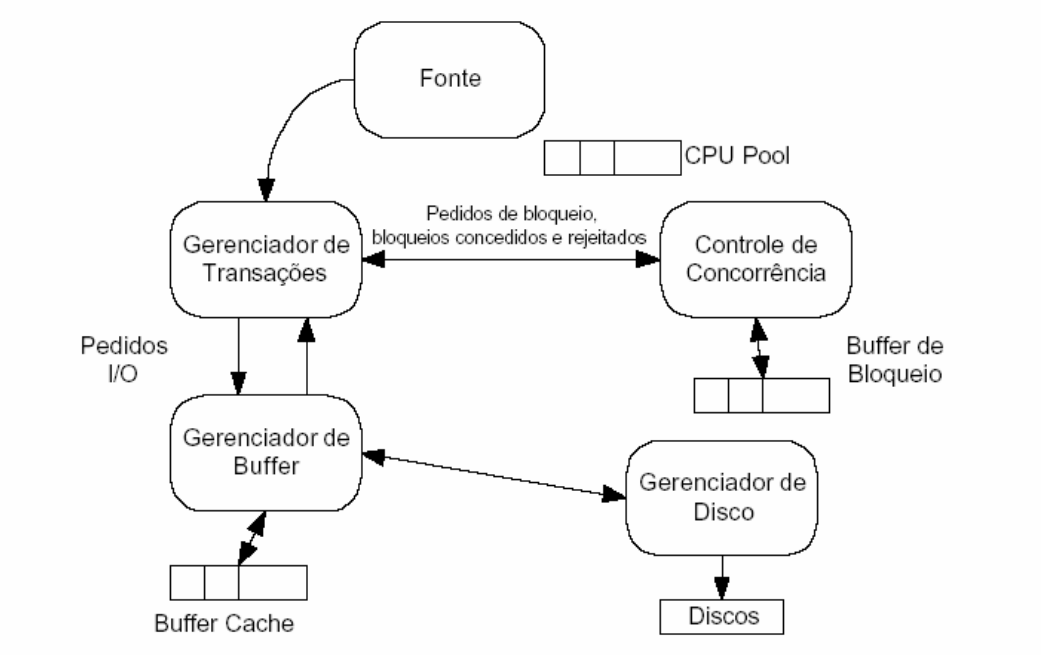

Figura 27 - Técnica Híbrida: Diagrama de blocos [PHATAK et al., 1995].

Segundo [GUPTA et al., 1997], esta abordagem híbrida demonstrou melhor desempenho do que os mecanismos de controle de concorrência baseados em bloqueio. Nas seções seguintes são resumidas as principais vantagens e desvantagens dos mecanismos de controle de concorrência estudados, bem como a sua maneira de resolver resolver conflitos, minimizar custo de transição de mensagens e minimizar a necessidade de reinício de transações. Também é apresentado como estes mecanismos são utilizados nos modelos de transações móveis.

4.4 Apreciação dos Mecanismos de Controle de Concorrência

No mecanismo **BO2F-TM** proposto por [JING et al., 1995], o bloqueio em duas fases é utilizado e por sua vez, concede bloqueios de leitura imediatamente, e adia bloqueios de escrita até o tempo de consolidação. A grande vantagem deste mecanismo é o fato da possibilidade do desbloqueio em local diferente da cópia no qual o bloqueio de leitura foi fixado. Com isto o algoritmo restringe o número de mensagens extras de desbloqueio de leitura, e como em uma unidade móvel as operações de leitura excedem em grande número as de escrita, aumenta o tempo de resposta das consultas feitas pelos clientes móveis. Este mecanismo utiliza número de versão de cópia para verificar suas atualizações.

Já a técnica híbrida explorada por [PHATAK et al., 1995] utiliza bloqueio em duas fases para resolver conflitos. Mas estes bloqueios estão vinculados ao tamanho do buffer de bloqueio existente. Se o número de *slots* do buffer for de número igual ao tamanho de itens do banco de dados, os conflitos serão resolvidos de forma otimista, seguindo apenas condições impostas a cada transação. Quanto menor a capacidade do buffer, maior será a utilização do bloqueio pessimista. A transação só é validada no momento da consolidação. Mas para isso todo o seu conjunto de leitura e escrita deve ser validado, só após esta validação o conjunto de escrita da transação é escrito no banco de dados.

Já o mecanismo de *timestamp otimista para transações móveis* TO-TM de [LEE et al., 2000] existe a necessidade de mais uma funcionalidade para garantir a seriabilidade das

operações. O que é percebido dentre os modelos é que os algoritmos que utilizam bloqueio, não utilizam somente o bloqueio pessimista mas sim um mecanismo híbrido com mecanismo otimista.

Os mecanismos UFO, OUFO, SCM e BM, utilizam o mecanismo de broadcast, onde ao invés dos clientes fazerem requisições ao servidor de banco de dados ocorre que o servidor periodicamente envia, em forma de broadcast, os dados para os clientes, sem a necessidade de requisições por parte destes. Utilizam o controle de concorrência na abordagem otimista, onde o processo de validação é feito por último, antes da fase de consolidação. O problema neste método é que a inconsistência de dados ocorre pelo fato dos clientes móveis fazerem a captação dos dados no "meio" do canal de *broadcast* a qualquer momento. O que pode ocasionar a leitura de dados obsoletos, pois estavam sendo atualizados por uma transação de atualização enquanto o processo de broadcast estava sendo feito com estes dados.

Neste ponto o mecanismo SCM para resolver tal problema, refaz o processo de *broadcast* dos itens de dados que foram atualizadas no servidor de banco de dados. O UFO gera um overhead quanto à resolução de conflitos, pois re-difunde todos os itens de dados. Enquanto que o SCM, mesmo utilizando o mesmo processo, re-difunde apenas os itens de dados que estavam em conflito. Por outro lado, o SCM gera um overhead para manter o gráfico de serialização das operações sempre atualizado.Já o mecanismo de broadcast multiversão envia várias versões dos itens de dados, fazendo com que menos transações necessitem reiniciar suas execuções

A seguir foi elaborada uma tabela com as principais vantagens de cada algoritmo apresentado bem como suas desvantagens.

Algoritmo	**Vantagens**	**Desvantagens**
BO2F-TM	Permite que uma operação de leitura desbloqueie um item de dados no *site* local e concede a execução da operação de leitura imediatamente (diminuindo o custo de transmissão de mensagens). E adia bloqueios de escrita até tempo	A performance é degradada em um ambiente onde a ocorrência de conflitos for alta.

	de consolidação. O algoritmo utiliza cópias replicadas de itens de dados para reduzir os custos de mensagens impostas pela mobilidade das transações. Ideal para aplicações com baixa contenção de dados	
TO-TM	Quando o escalonador recebe uma operação do gerenciador de transações, ele não decide se aceita ou não esta operação, e sim escalona imediatamente. Fazendo com que a performance aumente e o tempo de resposta diminua. Eficiente na detecção de conflitos.	Em um ambiente com alta contenção de dados este algoritmo não é adequado.
Técnica Híbrida	Nos sistemas móveis, uma transação pode ser pessimista enquanto a unidade móvel está executando em modo conectado, e será otimista quando a unidade móvel estiver desconectada. Segundo [GUPTA et al., 1997], este mecanismo demonstrou melhor desempenho do que os mecanismos de controle de concorrência baseados em bloqueio e *timestamp*.	Cada escrita é precedida por uma leitura do item de dados. Isto aumenta a troca de mensagens, aumentando assim o custo de comunicação.
UFO	Adequado para sistemas de difusão de dados, mecanismo de *broadcast*. Preserva a consistência dos dados. Aumenta o tempo de resposta das consultas feitas na unidade móvel. A implementação do algoritmo UFO é simples e não requer nenhuma alteração por parte dos clientes móveis.	Operações de leitura são executadas de forma não ordenada, ocasionando diversas operações de *re-broadcast,* o que degrada o desempenho do sistema.
OUFO	Busca reduzir o overhead de reinícios de transações, causado pelo resultado da operação de re-broadcast desenvolvido no algoritmo UFO, ordenando as operações de leitura. Além disso assegura a consistência de itens de dados armazenados localmente no cliente móvel. Sua performance é independente do número de clientes.	Com o objetivo de manter a ordem de execução das operações, se uma operação tiver que ser abortada, as operações dependentes da operação abortada em ordem de execução também precisam ser abortadas.
SCM	Faz o broadcast novamente apenas dos dados que estavam em conflito.	Apresenta *overhead* na atualização do gráfico de serialização.
Broadcast	É diminuído o número de abortos das transações	Há um aumento de volume no canal

Multiversão	neste modelo devido ao armazenamento local nas unidades móveis de várias versões dos itens de dados. Suporta as freqüentes desconexões das unidades móveis, uma vez que versões antigas ficam disponíveis no canal de *broadcast*.	de *broadcast* , com isso a probabilidade de aumento de tempo de resposta na unidade móvel se torna maior, a largura de banda pode não ser suficiente.

Tabela 8 - Vantagens e Desvantagens dos Algoritmos de Controle de Concorrência

A tabela a seguir sumariza as principais características dos algoritmos apresentados, quanto ao mecanismo utilizado, resolução de conflitos e custo de comunicação.

Algoritmo	**Mecanismo**	**Resolução de Conflitos**	**Custo de Comunicação**
UFO	Bloqueio em duas fases, abordagem otimista	Faz o *rebroadcast* de todos os dados (redifusão)	Caso a taxa de conflitos for alta, a troca de mensagens será alta também, aumentando o custo de comunicação do algoritmo.
OUFO	Bloqueio em duas fases abordagem otimista	Faz o *rebroadcast* de todos os dados (redifusão), mas em menor número devido a ocorrência de conflitos ser menor.	Igualmente ao algoritmo UFO.
SCM	Bloqueio em duas fases, abordagem otimista	Faz apenas o rebroadcast dos itens de dados em conflito.	Aumenta o custo de comunicação na troca de mensagens para manter os gráficos atualizados.
Broadcast Multiversão	Bloqueio em duas fases abordagem otimista	Utiliza versões antigas dos itens de dados, diminuindo consideravelmente a taxa de ocorrência de conflitos.	Como diminui a taxa de conflitos, diminui também a troca de mensagens. Portanto possui custo de comunicação baixo.
BO2F-TM	Bloqueio em duas fases, abordagem otimista	Há troca de mensagens de permissões com o gerenciador de transações que decide se um pedido de bloqueio de escrita pode ser adiado até o tempo de consolidação	Coordenador armazena mensagens de respostas e informações de desbloqueio dos *sites*, com isso diminui a troca de mensagens e o custo de comunicação se torna menor
TO-TM	Ordenação por *timestamp,* abordagem otimista.	Emprega várias estruturas de dados: uma tabela contendo os nomes e os *timestamps* das transações ativas; um conjunto consistindo das operações de leituras que foram escalonadas da transação móvel, E outro conjunto das escritas escalonadas. Antes da consolidação faz testes de conflitos utilizando comparações de valores armazenados nestas estruturas.	
Técnica	Mecanismo	Prioriza acessos pessimistas, que é o	Cada escrita é precedida

Híbrida	Híbrido, utilizando a abordagem pessimista com a otimista.	adequado para ambientes com muitos conflitos.	por uma leitura dos itens de dados, isto encarece a troca de mensagens, encarecendo o custo de comunicação.

Tabela 9 - Apreciação dos Algoritmos de Controle de Concorrência

4.5 Mecanismos de Controle de Concorrência Utilizados pelos Modelos de Transações Móveis

Os métodos de controle de concorrência otimistas são os mais utilizados, pois são mais flexíveis, por considerarem a quase inexistência de conflitos. Como as unidades móveis em certos momentos podem estar operando em modo desconectado, e as transações móveis têm a necessidade de continuar a computação, os métodos pessimistas são menos utilizados, pois podem levar a *deadloks* , com freqüência

4.5.1 Modelo MDSTPM

Esta abordagem faz a unidade móvel ser parte dos múltiplos bancos de dados durante a sua conexão com o respectivo nó coordenador, a MSS. Neste modelo é assumida a existência de um MDSTPM no topo de um SGBD, que pode escalonar e coordenar a execução das transações globais conforme o interesse das unidades móveis. Cada transação global ou distribuída deverá encontrar o coordenador, que efetuará todas configurações do ambiente para a execução. Este modelo é mais indicado para redes do tipo *ad-hoc*, onde todos os participantes são totalmente autônomos.

Todos os SGBDs envolvidos deverão implementar o mecanismo de controle de concorrência de *bloqueio em duas fases*, de acordo com [YEO & ZASLAVSKY, 1994]. No entanto para forçar a ordem de execução das transações globais uma abordagem para solucionar este problema é proposta, a obtenção de um *timestamp* lógico para cada transação global. Este *timestamp* lógico é armazenado no sistema local de banco de dados, onde a transação global é forçada a obter um "*timestamp* lógico" em sua chegada. Tal ação deverá causar um conflito adicional entre subtransações globais desta forma mantendo sua ordem de execução.

A coordenação da execução das transações móveis neste modelo é feita de forma centralizada. Para gerenciar transações globais o modelo MDSTPM implementa o mecanismo de concorrência *de bloqueio em duas fases* [YEO & ZASLAVSKY, 1994] e [ALVARADO et al., 2001]. Com a utilização deste mecanismo de controle de concorrência, apesar de solucionar de forma eficiente abortos em cascata, ele apresenta algumas desvantagens como muitas trocas de mensagens entre a unidade móvel e o seu respectivo *site* coordenador. Além desta desvantagem, o tempo de bloqueio dos dados é indefinido devido às desconexões.

4.5.2 Modelo Canguru

Como no modelo MDSTPM as propriedades ACID não são afetadas pela mobilidade pois a execução das transações é de responsabilidade do SGBD localizado em uma estação fixa. Entretanto, como transações são requisitadas pelas unidades móveis, a mobilidade e as desconexões devem ser gerenciadas.

No TC para suportar a mobilidade das unidades móveis e as freqüentes desconexões , o Agente de Acesso aos Dados (AAD) segue o movimento da unidade móvel através de uma lista de links de todas as MSS que foram coordenadoras de transações móveis. Esta lista será utilizada no caso de abortos em cascata. Existem também estruturas (tabela de status das transações e tabela de log local) que armazenam informações das transações móveis como: identificador das transações globais, estado (ativa, consolidada, abortada), o identificador da transação Joey, subtransações que foram incluídas nas transações Joey, transações compensadas (se existirem), etc.

Ao contrário do mecanismo de MDSTPM, no modelo TC a coordenação das transações móveis é distribuída ao longo de todas as estações base, MSS, que a unidade móvel visita. Desta forma conclui-se que o modelo TC trata da natureza móvel das transações, e não somente com respeito as desconexões. A coordenação distribuída reduz o custo de comunicação durante a execução, entretanto nos casos de abortos em cascata a comunicação é aumentada drasticamente.

4.5.3 Modelo Clustering

Para gerenciar o isolamento, *Clustering* propõe nova tabela para soluções de conflitos. Este modelo utiliza o mecanismo de controle de concorrência de bloqueio em duas fases**, *strict two phase locking*** [ALVARADO et al., 2001]**,** e propõe quatro tipos de bloqueio que correspondem às operações fracas e estritas (F_Leitura(x), F_Escrita(x), R_Leitura(x), R_Escrita(x)). Uma operação de F_Leitura(x) em um C_i é traduzida para uma leitura fraca em uma cópia local de x, ou seja, uma F_Leitura(x^f_i) da mesma forma uma R_Leitura(x) em R_Leitura(x^f_i). É importante salientar que esta atualização só pode se tornar visível para transações rígidas quando a transação fraca obter um *commit* global.

	F_Leitura(x_j)	R_Leitura(x_j)	F_Escrita(x_j)	R_Escrita(x_j)
F_Leitura(x_i)			**conflitam**	**conflitam**
R_Leitura(x_i)				**conflitam**
F_Escrita(x_i)	**conflitam**		**conflitam**	**conflitam**
R_Escrita(x_i)	**conflitam**	**conflitam**	**conflitam**	**conflitam**

Tabela 10 - Relação de Conflitos

A Tabela 10 mostra a nova tabela de conflitos que é proposta pelo modelo *Clustering*. Transações fracas liberam seus bloqueios para *commit* local e transações rígidas para *commit* (global). Este modelo requer um Gerenciador de Transações na própria unidade móvel para fornecer a execução das transações localmente, controle de concorrência, gerenciamento de *logs* e recuperação. Para manter a consistência, um nível de inconsistência é permitido para as versões fracas na UM, este nível é obtido limitando o número de *commits* locais, o número de transações que podem operar em cópias inconsistentes.

4.5.4 Modelo Baseado em Semântica

A serializabilidade é mantida através do protocolo de controle de concorrência de bloqueio em duas fases otimista, quando as transações locais têm acesso ao *cache* de fragmentos, [WALBORN & CHRYSANTHIS, 1995].

Este modelo faz *commit* local, na unidade móvel, em modo desconectado. Por este motivo a abordagem otimista é a mais adequada neste caso. E faz *commit* global, no servidor de banco de dados, com mecanismos de reconciliação, juntando os fragmentos.

4.5.5 Modelo Pro-Motion

Um *compact* pode prover uma operação de leitura rígida (strict read), a qual somente completa se o item for bloqueado e, o método leitura fraca (relaxed read) o qual retorna um valor apesar da condição do item de dado. Como o modelo *Pro-motion* usa a arquitetura aninhada particionada (*nested split*), o protocolo de bloqueio em transações aninhadas obedecem as seguintes regras:

- antes de realizar uma operação no item de dado x, denotado por $\mathbf{op_i\ [x]}$, uma transação deve obter o bloqueio deste item, denotado por $\mathbf{bl_i\ [x]}$;
- a transação só obtém o bloqueio de bl_i [x] se nenhuma outra transação T_j no sistema possuir um bloqueio conflitante bl_j [x]; caso contrário, a transação é colocada em modo de espera;
- ao abortar, todos os bloqueios de uma transação são liberados;
- uma transação só se confirma se todas as suas filhas já terminaram (confirmadas ou abortadas);
- ao se confirmar, todos os bloqueios de uma transação passam para sua mãe.

Quando a unidade móvel se reconecta à rede, começa novamente a sincronização. Esta sincronização busca as mudanças executadas na unidade móvel, para o servidor e requer alguns estágios, dependo da validade dos *compacts* contidos no *cache* da unidade móvel. São feitas tentativas para incorporar as mudanças oriundas de todas as transações consolidadas localmente na unidade móvel. Porém, pode não ser possível consolidar toda transação ilegível ao servidor. O procedimento contingente para cada transação que não pode executar uma operação de *commit* no final, é executar uma compensação.

4.5.6 Modelo Reporting

Segundo [CHRYSANTHIS & RAMAMRITHAN, 1993], este modelo analisa transações aninhadas e transações aninhadas abertas, mostrando as limitações de seus ambientes móveis. Neste modelo é proposta a transação aninhada aberta que suporta atomicidade, transações não-compensáveis e dois tipos adicionais: reporting e co-transações.

Transações aninhadas fechadas consolidam de baixo para cima até a raiz. Conseqüentemente, uma subtransação aninhada começa depois de seu pai e termina antes. E a consolidação da subtransação é condicionada à consolidação de seu pai. O aninhamento aberto é flexível quanto à restrição de atomicidade do nível superior da transação aninhada fechada, onde os resultados só podem ser vistos após a consolidação da transação pai. A transação aninhada aberta permite que seus resultados parciais sejam observados fora da transação [ÖSZU 2001]. Enquanto em execução, as transações podem compartilhar os resultados parciais.

Uma transação móvel é estruturada como um conjunto de transações, algumas das quais são executadas na unidade móvel. É considerado que, limitações na unidade móvel fazem necessário o uso de uma estação de suporte a mobilidade (MSS), por exemplo, para armazenar parte do estado da computação ou executar parte da computação.

4.5.7 Modelo Two-Tier

É assumido assim que a unidade que originou a transação repassa as atualizações de réplica a todas as réplicas "escravas" depois que a transação mestre consolidar. E também envia uma transação escrava para cada unidade de réplica escrava. Atualizações de "unidade escrava" são **pré-ordenadas** para assegurar que todas as réplicas convergem ao mesmo estado final.

Se o registro de marca de tempo ou *timestamp* da réplica for mais atual que um registro de marca de tempo da atualização de réplica, a atualização (no caso mais antiga) é ignorada. Alternativamente, cada unidade mestre envia atualizações de réplica para réplicas escravas na ordem da seqüência do *commit*.

Um nó que quer atualizar um objeto deve ser conectado ao dono de objeto e deve ser participante em uma transação atômica com o dono. Como com sistemas não flexíveis, sistemas de mestre-flexíveis não têm nenhum fracasso de reconciliação; pois, conflitos são resolvidos esperando pelo *deadlock*. Se for ignorada a mensagem de demora, a taxa de *deadlock* para um sistema de replicação de mestre-flexível é semelhante a um sistema de um nó único com taxas de transação mais altas.

Transações de mestre flexível operam em cópias de mestre de objetos. Os nós que contém tempos para muitas transações concorrentes de mestre, são calculados em aproximadamente nó^2 (tempos de nós ao quadrado) de tempo para muitas transações de atualização de réplica. As transações de atualização da réplica não importam, elas são transações de trabalho de *background* (fundo). Eles podem abortar e podem reiniciar sem afetar o usuário. Assim o fator principal é o freqüente *deadlock* das transações de mestre. Este é o melhor comportamento da replicação de grupo-relaxado. A replicação de mestre flexível envia menos mensagens durante a transação básica e assim completa mais depressa. Mas, todos estes esquemas de replicação têm como aborrecimento à taxa de *deadlock* ou reconciliação quando há muitos nós.

De acordo com [THOMAS, 1979] o método utilizado para que transações possam ser executadas as atualizações, inclusive com concorrência, é a **pré-ordenação,** *timestamp*. Na pré-ordenação um valor é substituído por um valor mais novo. Se o valor atual do objeto já tem uma marca de tempo (*timestamp*) maior que a marca de tempo da atualização, a atualização está descartada.

4.5.8 Modelo Prewrite

O algoritmo de controle de concorrência utilizado neste modelo é algoritmo de bloqueio em ***duas fases*** [MADRIA & BHARGAVA, 1998]. Na **primeira fase**, o controle de concorrência para o controle de operações *prewrite* e *pre-read* é executado no GT, gerenciador de transação, na estação base. Na **segunda fase**, o controle de concorrência para o controle das operações (para acessar os valores de escrita, gravação) é executado no GD, gerenciador de dados.

Desde que os valores de *prewrite* são visíveis depois do *pre-commit*, o tipo de bloqueio mantido pela operação de *prewrite* é convertido em tipo de bloqueio para operações *write*, não ocasionando nenhum conflito de bloqueio. O bloqueio adquirido para uma operação de *prewrite* não é liberado depois do *pre-commit* devido ao *bloqueio de duas fases* não permitir que uma transação adquira bloqueios depois de ter liberado alguns.

Neste modelo são usados quatro tipos de bloqueio:

- *read-lock* para operações de *read* (leitura);
- *pre-read-lock* para operações de *pre-read;*
- *write-lock* para operações de *write*; e
- *prewrite-lock* para operações de *prewrite;*

	read-lock	pre-read-lock	prewrite-lock	write-lock
read-lock				x
pre-read-lock			x	x
prewrite-lock		x	x	
write-lock	x			x

Tabela 11- Tabela de Conflitos dos Tipos de Bloqueios [MADRIA & BHARGAVA, 1998a].

Um bloqueio do tipo *prewrite-lock* ocasiona conflito com outros bloqueios *pre-read-lock* e *prewrite-lock*, entretanto não conflita com bloqueios de *read-lock* e de *write-lock*, como demonstrado na Tabela 11 (um "X" indica conflito entre as operações). Um bloqueio de *prewrite-lock* não pode ser liberado depois do *pre-commit,* pois a transação ainda precisa do bloqueio *write-lock* para as atualizações finais. Um *prewrite-lock* adquirido por uma transação que executou um *pre-commit* é convertido para um *lock-write* fazendo com que nenhuma transação possa adquirir este bloqueio conflitante. Uma vez um bloqueio do tipo *prewrite-lock* é atualizado para um bloqueio do tipo *write-lock* , a mesma transação não pode adquirir qualquer outro bloqueio. Entretanto, bloqueios do tipo *pre-read-locks* podem ser adquiridos por outras transações para acessar valores *prewrite*. Segundo [MADRIA & BHARGAVA, 1998], o protocolo de bloqueio utilizado está apresentado no algoritmo que segue.

Pre-read-Lock(x): concede o pedido *pre-read-lock* para a transação T em X se nenhuma outra transação detém um *prewrite-lock* em X;

Read-lock (x): concede o pedido *read-lock* para a transação T em X se nenhuma outra transação detém um write-lock em X.;

Prewrite-lock (x): concede o pedido *prewrite-lock* para a transação T em X se nenhuma outra transação detém um *prewrite-lock* ou *pre-read-lock* em X.;

Write-lock (x): atualiza um *prewrite-lock* em X obtido por uma transação T para *write-lock* se

se a fila de *write-lock* em X estiver vazia **então**
se a transação T é *pre-commited* e não ha outra transação que detenha um *read-lock* ou *write-lock* em X **então**
Converta *prewrite-lock* para *write-lock*;
fim se;
senão
coloca a transação T na fila de espera *write-lock* para X;
fim se;

1. Uma transação T é enviada para a unidade móvel.
2. A UM analisa a transação T para verificar suas requisições de *read*, *pre-read* e *write* (é assumido que a UM tem algumas capacidades de servidor);
   /*a parte de pré-commit da transação é executada na Estação Base MSS */
3. **Se** (a transação T possui operações de *pre-read, read* e *write*) **então**
   Para todas operações ***pre-read***, ***reads*** e ***writes*** ∈ T
   A UM envia requisições para a MSS para os bloqueios de *pre-read*, *read* e *prewrite* requeridos;
   Depois que a MSS adquire os bloqueios necessários, ela retorna os valores de *pre-read* e *read* para a UM (somente para aqueles valores de *read* e *pre-read* que não estiverem disponíveis localmente na UM, neste caso bloqueios precisam ser adquiridos)

   Para todos ***writes*** ∈ T
   Anuncia todos os valores do *prewrite* na UM (sem esperar por *prewrite-locks*);
   Armazena os logs de *prewrite* na UM;
   Grava os registro de log do *pre-commit* e ;

Envia os valores *prewrite*, log *prewrite* e outros registros de logs para a MSS;

Se a MSS adquire todos os *prewrite-locks* para todos os itens de dados que foram anunciados pelo *prewrite* **então**

MSS aceita os valores *prewrite* e logs;

MSS atualiza o *prewrite-lock* para *write-lock* /**fazendo com que nenhuma transação possa acessar este dado*/

Senão valores *prewrite* e logs correspondentes são descartados pela UM

fim se;

fim;

4. Para cada *Prewrite* anunciado /**após o pre-commit* o algoritmo é executado na MSS*/

Atualiza todos aqueles objetos no banco de dados com os quais foram anunciados no *prewrite*;

Armazene os registros de logs necessários;

O modelo de transações móveis *Prewrite* proporciona um aumento na disponibilidade dos dados devido às transações poderem ser executadas concorrentemente durante desconexões ambos nas UMs e nas MSS. O modelo permite a execução da transação alternando entre a UM e a MSS para atualizações no banco de dados. Assim, reduzindo o custo de computação [MADRIA & BHARGAVA, 1998a], [MADRIA & BHARGAVA, 1998b].

4.5.9 Classificação dos Mecanismos de Controle de Concorrência de Acordo com a Taxonomia Proposta

Como já estudado no capítulo 2, a arquitetura de implementação dos sistemas gerenciadores de banco de dados móveis, SGBDM, influi diretamente nos modelos de execução, gerenciamento de transações e conseqüentemente nos mecanismos de controle de concorrência. Para uma arquitetura totalmente distribuída e simétrica, modelo ideal para redes *ad-hoc*, todos os controles da execução da transação e mecanismos de concorrência devem ficar por conta de cada SGBDM participante. Nos

atuais modelos de computadores móveis pessoais este tipo de modelo não é indicado, visto que possuem baixo poder de processamento, pouca memória e pouco tempo de autonomia de bateria. Já em um modelo cliente/servidor estendido, caso de muitas implementações dos SGBDM comerciais existentes, a infra-estrutura de rede é um elemento essencial para seu funcionamento.

Os modelos apresentados e estudados propõem soluções através de mecanismos de controle de concorrência em um ambiente com uma infraestrutura existente. Partem do princípio que as unidades móveis estarão, em algum momento, conectadas à rede fixa. Desta maneira todos os modelos podem ser classificados como integrantes da arquitetura Cliente/Servidor estendido, proposto por [PITOURA & SAMARAS, 1998].

Os tipos de transações utilizados nos modelos estudados utilizam os mesmos tipos de transações dos SGBD distribuídos, e serão importantes no mecanismo de controle de concorrência adotado. No modo conectado o SGBDM é mais um SGBD participante de um ambiente distribuído, utilizando-se das tradicionais técnicas dos SGBDD, e como neste ambiente a ocorrência de conflitos é muito grande os métodos enquadrados na abordagem pessimista são mais indicados.

Já no modo desconectado, a grande preocupação está na sincronização dos dados atualizados com suas respectivas cópias primárias no SGBD da rede fixa, onde utilizamos os mecanismos de controle de concorrência na abordagem otimista, pois o modo desconectado tem como característica a quase inexistência de conflitos.

A Tabela 12 classifica os modelos de transações móveis quanto o seu tipo de arquitetura, tipo de transações e principalmente classifica estes modelos quando a taxonomia proposta.

Modelo	**Arquitetura**	**Tipo de Transação**	**Mecanismo de Controle de Concorrência**
MDSTPM	Arquitetura ad-hoc. Onde as estações operam de maneira autônoma.	Transações Globais e Locais	Utiliza o mecanismo de bloqueio em duas fases com utilização de um timestamp lógico para as transações

	Utilizam o mecanismo de RPC para comunicação entre as unidades. .		globais, se enquadra na abordagem pessimista híbrida.
Canguru	Arquitetura Cliente/Servidor Estendido	Transações Aninhadas Abertas	Mecanismo de bloqueio em duas fases, com abordagem pessimista
Clustering	Ambiente de sistemas totalmente distribuído. Banco de Dados é dividido em *clusters*	Transações fracas e transações rígidas	Mecanismo de bloqueio em duas fases na abordagem otimista
Semântico	Arquitetura cliente/servidor estendido	Transações de Longa Duração	Mecanismo de bloqueio em duas fases na abordagem otimista
Prewrite	Arquitetura cliente/servidor estendido	Transações Aninhadas Abertas	Mecanismo de bloqueio em duas fases estendido, e novas tabelas de conflitos foram propostas
Pro-Motion	Arquitetura cliente/servidor estendido	Transações Aninhadas Abertas	Mecanismo de bloqueio em duas fases
Reporting	Sistema especial de vários bancos de dados	Transações Aninhadas Abertas, não-compensáveis, "reporting" e co-transações	Mecanismo de bloqueio em duas fases
Two-Tier	Ambiente de sistemas totalmente distribuído.	Transações Base e Tentativas	Pré-ordenação (*TimeStamp*) – Pessimista

Tabela 12 - Mecanismos de Controle de Concorrência dos Modelos de Transações Móveis

Os modelos de transações móveis estudados, em virtude das características do ambiente móvel, tiveram que propor diferentes tipos de transações, com novas abordagens em relação às propriedades ACID, com objetivo de proporcionar melhores formas de execução das transações neste ambiente.

Em virtude das alterações destas propriedades, mecanismos de controle de concorrência para ambientes móveis foram implementados, buscando resolver os problemas enfrentados neste ambiente. Cada modelo de transação móvel apresenta soluções quanto ao mecanismo de controle de concorrência utilizado (podemos observar pela Tabela 12 os tipos de mecanismos utilizados por cada modelo). Neste trabalho buscamos descrever

os modelos e estudá-los com a finalidade de classificar os mecanismos de controle de concorrência e estabelecer uma taxonomia destes mecanismos.

5 Conclusões e Perspectivas de Trabalhos Futuros

Este trabalho concentrou esforços sobre os problemas e as características de banco de dados móveis em um ambiente de computação móvel, suas implicações e restrições. Foram abordadas as alterações das propriedades ACID, propostas para garantir a consistência dos dados, em virtude dos diversos problemas da computação móvel, como por exemplo, a mobilidade, portabilidade, desconexões e baixa largura de banda.

Foram expostos modelos de transações móveis que tentam suprir as necessidades do banco de dados na arquitetura de computação móvel, assim como suportar a desconexão, a reintegração e a mobilidade. Foram feitas tabelas de classificação quanto ao suporte às propriedades ACID, por cada modelo apresentado.

Mecanismos de Controle de concorrência adequados para suportar as transações móveis foram apresentados e além disto, uma proposta de taxonomia para estes mecanismos de controle de concorrência foi desenvolvida, permitindo assim uma melhor classificação dos mecanismos para o ambiente móvel.

Percebemos, apartir de algumas classificações feitas, que os algoritmos utilizados pelos modelos de transações móveis que se enquadram na abordagem pessimista, principalmente utilizando bloqueio, ocasionam um *overhead* na rede devido à quantidade de mensagens enviadas, degradando o desempenho do sistema.

Já algoritmos utilizando a abordagem otimista apresentam melhores resultados, pois permitem que itens de dados sejam acessados sem muitas trocas de mensagens, em ambientes onde as taxas de ocorrências de conflitos sejam baixas, proporcionando um melhor desempenho no sistema. Ressaltando que esta abordagem não é adequada para ambientes onde é freqüente a ocorrência de conflitos, podendo ocasionar *deadlock*.

No entanto, de acordo com diversas pesquisas, a abordagem híbrida seria mais apropriada, em virtude de sua flexibilidade, pois permite utilizar a abordagem

pessimista estando às unidades conectadas a rede, onde a taxa de conflitos é alta, e a otimista para ambientes onde a ocorrência de conflitos é quase inexistente, no caso quando as unidades estiverem desconectadas da rede.

Para trabalhos futuros, tendo em vista que os fatores primordiais para o controle de concorrência em ambientes móveis são a minimização de mensagens trocadas, a garantia de consistência e a performance de execução do algoritmo adotado, é sugerida análise comparativa de desempenho e performance (tempo de resposta das consultas, quantidade de reinícios de transações, avaliação de abortos das transações) dos modelos de transações móveis utilizando mecanismos de controle de concorrência classificados na abordagem híbrida.

Através desta análise poderíamos proporcionar uma indicação baseada em experimentos sobre quais mecanismos contribuiriam para uma melhor performance na utilização de bancos de dados móveis.

Referências Bibliográficas

[ACHARYA et al., 1995a] ACHARYA, A; ALONSO, R; FRANKLIN, M. J; ZDONIK, S. **"Broadcast Disks: Data Management for Asymmetric Communications Environments"**. In proceedings of the ACM SIGMOD International Conference on Management of Data, Junho, 1995.

[ACHARYA et al., 1995b] ACHARYA, A; FRANKLIN, M. J; and ZDONIK, S. **"Dissemination-Based Data Delivery Using Broadcast Disks"**. IEEE Personal Communications. Dezembro, 1995.

[ALONSO & KORTH, 1993] ALONSO, R; KORTH, H. **"Database system issues in nomadic computing"**. In proceedings of the 1993 SIGMOD Conference, Washington, D.C., 1993.

[ALVARADO et al., 2001] ALVARADO, P; RONCANCIO, C; ADIBA, M. **"Mobile Transaction Supports for DBMS: Na Overview"**. LSR-IMAG Laboratory, 2001.

[ARAÚJO & FERREIRA, 2000] ARAÚJO, L. V; FERREIRA, J. E. **"Cache Semântico para Computação Sem Fio Baseado na Abstração de Composição de Dados".** WorkSIDAM, Workshop de Sistemas de informação Distribuída de Agentes Móveis, p. 83-89, São Paulo, Outubro, 2000.

[AU et al., 2001] AU, M; CHAN, E; LAM; K. **"concurrency control for mobile systems with data broadcast"**. Journal of Interconnection Networks, Vol. 2, No. 3, pg. 253-267. Hong Kong, 2001.

[BADRINATH et al., 1993a] BADRINATH, B. R; ACHARYA, A; IMIELINSKI, T. "**Impact of mobility on distributed computations**". Operating Systems Review, Abril 1993.

[BADRINATH et al., 1993] BADRINATH, R; BAKRE, A; IMIELINSKI, T. "**Handling Mobile Clients: A Case for Indirect Interaction".** In Proceedings of the 4th Workshop on Workstation Operation Systems, Aigen, Austria, Outubro, 1993.

[BADRINATH & IMIELINSKI, 1992] BADRINATH, R; IMIELINSKI, T. **"Replication and mobility"**. Proceedings of the second Work-Shop on the Management of Replicated Data, Dezembro 1992.

[BALDWIN et al., 1996] BALDWIN, F; MCVOY, S; STEINFIELD, C. "**Convergence: Integrating Media, Information & Communication"**. Thousand Oaks and London: Sage Publications, 1996.

[BARBARA, 1997] BARBARA, D. **"Certification Reports: Supporting Transactions in Wireless Systems"**. In Proceedings of IEEE International Conference on Distributed Computing Systems, pp.466–473, 1997.

[BERNSTEIN et al., 1987] BERNSTEIN A; GOODMAN, N; HADZILACOS, V. "**Concurrency Control and Recovery in Database Systems**". Addison-Wesley Series in Computer Science , 1987.

[BREITBART et al., 1990] BREITBART, Y; SILBERSCHATZ, A; THOMPSON R. **"Reliable transaction management in a multidatabase system"**. Proceedings of the ACM SIGMOD international conference on Management of data, ACM Press, Pg. 215-224, Atlantic City, New Jersey, United States, 1990.

[BREITBART et al., 1992] BREITBART, Y; GARCIA-MOLINA, H; SILBERSCHATZ, A. **"Overview of multidatabase transaction management"**. VLDB Journal, 2, 1992.

[BURGER et al., 1997] BURGER, A; KUMAR, V; HINES, M. **"Performance of Multiversion and Distributed Two-Phase Locking Concurrency Control Mechanisms in Distributed Databases"**, Information Sciences, vol. 96, pg. 129-152, 1997.

[CAREY & LIVNY, 1991] CAREY, M; LIVNY, M. **"Conflict detection tradeoffs for replicated data"**. Source ACM Transactions on Database Systems (TODS), New York, USA, Vol. 16, Pg. 703-746. Dezembro, 1991.

[CHESS et al., 1995] CHESS, D; GROSOF, B; HARRISON, D. Levine; PARRIS, C; TSUDIK, G. "**Itinerant Agents for Mobile Computing"**. IEEE Personal Communications, 2(5), Outubro 1995.

[CHRYSANTHIS, 1991] CHRYSANTHIS, P. K. **"ACTA, A Framework for Modeling and Reasoning about Extended Transactions".** PhD thesis, Departament of Computer and Information Science, University of Massachusetts, Amherst, September 1991.

[CHRYSANTHIS, 1993] CHRYSANTHIS, P. **"Transaction processing in mobile computing environments"**. Proceedings of IEEE Workshop on Advances in Parallel and Distributed Systems, 1993.

[CHRYSANTHIS & RAMAMRITHAN, 1993] CHRYSANTHIS, K; RAMAMRITHAM, K. **"Synthesis of Extended Transaction Models Using ACTA".** Technical Report 93-05, University of Pittisburg, 1993.

[CHRYSANTHIS, 1995] CHRYSANTHIS, P. "**Supporting Semantics based Transaction Processing in Mobile Database Applications"**. In 14th IEEE Symposium on Reliable Distributed Systems. September, 1995.

[CHUNG et al., 2003] CHUNG, Il; BHARGAVA, B.; MAHOUI, M.; LILIEN, L. **"Autonomous Transaction Processing Using Data Dependency in Mobile Environments"**. The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03), May , San Juan, Puerto Rico, 2003.

[CLARK & DEMIR, 2003] CLARK, L; DEMIR, O. **"Transaction Management in Mobile Distributed Databases"**. June, 2003.

[DAS & KAI, 2001] DAS, A; KAI, S. **"Tradeoff between Client and Server Transaction Validation in Mobile Environment".** Proc. Int'l Symp. Database Eng. & Applications, pp. 265-272, 2001.

[DATE, 2000] DATE, C.J. **"Introdução a Sistemas de bancos de Dados"**. Rio de janeiro: Campus, 2000.

[DIRCKZE & GRUENWALD, 1998] DIRCKZE, R; GRUENWALD, L. **"A Toggle Transaction Management Technique for Mobile Multidatabases"**. Proceedings of the ACM – CIKM, 1998

[DUNHAM & HELAL, 1995] DUNHAM, M; HELAL, A. **"Mobile computing and databases: Anything new? "**. ACM SIGMOD Record, 24(4), 1995.

[DUNHAM et al., 1997] DUNHAM, M; HELAL, A; BALAKRISHNAN, S. **"A mobile transaction model that captures both the data and a movement behavior".** Mobile Networks and Applications 2 , páginas 149-162, 1997.

[DUNHAM & KUMAR, 1998] DUNHAM, M; KUMAR, V. **"Location Dependent Data and its Management in Mobile Databases"**. Proceedings of the Ninth International Workshop on Database and Expert Systems Applications, DEXA Workshop, pg 414-419, August, 1998.

[DUNHAM & KUMAR, 1999] DUNHAM, M; KUMAR, V. **“Impact of Mobility on Transaction Management ”.** In Proc. of MobiDE /Mobicom99 Workshop, Seattle, WA, pages 14-21. ACM, 1999.

[ELMAGARMID et al., 1995] ELMAGARMID, A. K; JING, J; BUKHRES, A. **“An Efficient and Reliable Reservation Algorithm for Mobile Transactions.”** In Proceedings of the 4th International Conference on Information and Knowledge Management (CIKM'95), 1995.

[FERREIRA & FINGER, 2000] FERREIRA, E; FINGER, M. **“Controle de concorrência e distribuição de dados: a teoria clássica, suas limitações e extensões modernas”**. São Paulo, Escola de Computação 2000 - IME-USP, 2000.

[FORMAN, 1994] FORMAN, G. **“The Challenges of Mobile Computing”**. UW CSE Technical Report, March, 1994.

[GRAHAM et al., 1995] GRAHAM, P; BARKER, K; REZA-HADAEGH, A. **“Disconnected Objects: Reconciliation in a Nested Object Transaction Environment"**. Url = citeseer.nj.nec.com/4896.html.

[GRAY et al., 1996] GRAY, J; HELLAND, P; O’NEIL, P. ; and SHASHA, D. **“The dangers of replication and a solution**”. In Proceedings of ACM-SIGMOD International Conference on Management of Data, páginas 173–182, Montreal, Canadá, 1996.

[GRUENWALD & BANIK, 2001] GRUENWALD, L; BANIK, S. **"A Power-Aware Technique to Manage Real-Time Database Transactions in Mobile Ad-Hoc Networks ",** 4th International Workshop on Mobility in Database and Distributed Systems, part of the International Conference on Database and EXpert systems Applications (DEXA), September 2001.

[GUPTA et al., 1997] GUPTA, R; HARITSA, J; RAMAMRITHAM, K. **"Revisiting Commit Processing in Distributed Database Systems".** In Proc. of the ACM SIGMOD, 1997.

[HARITSA et al., 1997] HARITSA, J; RAMAMRITHAM K; GUPTA, R. "**Characterization and Optimization of Commit Processing Performance in Distributed Database Systems** ". In Proc. of the ACM SIGMOD, Intl. Conf. on Management of Data, Tucson, Arizona, May 1997.

HOLLIDAY, J; AGRAWAL, D; EL ABBADI, A. **"Exploiting Planned Disconnections in Mobile Environments"**. In Proceedings, 10th IEEE Workshop on Research Issues in Data Engineering (RIDE2000), pages 25--29, Feb. 2000.

[IMIELINSKI & BADRINATH, 1994] IMIELINSKI, T. ; BADRINATH, B. R. **"Mobile Wireless Computing: Solutions and Challenges in Data Management".** Technical report, DCS-TR-296 and WINLAB TR-49, Rutgers University, Dept. of Comp. Sciences, 1993.To appear in Communications of the ACM, 1994.

[IMIELINSKI & BADRINATH, 1992] IMIELINSKI, T; BADRINATH, B. R. **"Querying in highly mobile distributed environments".** Proceedings of the 18 th VLDB Conference, December 1992.

[IMIELINSKI et al., 1994] IMIELINSKI, T; VISWANATHAN, S; BADRINATH, B. R. "**Energy efficient indexing on air".** Proceedings of the ACM SIGMOD international conference on Management of data, pag. 25-36, 1994.

[JING et al., 1995] JING, J; BUKHRES, O; ELMAGARMID, A. **"Distributed Lock Management for Mobile Transactions". Proc. 15th IEEE** International Conference on Distributed Computing System, Vancouver, British Columbia, Canadá, pp.118-125, 1995.

[JOSEPH et al., 2000] JOSEPH, S; HATTORI, M; KASE, N. **"Efficient Search Mechanisms for Learning Mobile Agent Systems"**. Efficient Search Mechanisms For Learning Mobile Agent Systems. Concurrency: Practice and Experiment. In Press, 2000.

[KISTLER & SATYANARAYANAN, 1992] KISTLER, J; SATYANARAYANAN, M. **"Disconnected Operation in the Coda File System**". ACM Transactions on Computer Systems, Fevereiro 1992.

[KUMAR & DUNHAM, 1998] KUMAR. V; DUNHAM. M. **"Defining Location Data Dependency, Transaction Mobility and Commiting."**. Technical Report. Department of Computer Science and Engineering. Dallas, 1998.

[KUMAR et al., 2002] KUMAR, V; PRABHU, N; DUNHAM, M; SEYDIM; A. **"TCOT-A Timeout-Based Mobile Transaction Commitment Protocol"**. IEEE Transactions on Computers 51(10): 1212-1218, 2002.

[LAM et al., 1999a] LAM, K; AU, M; CHAN, E. **"Broadcast of Consistent Data to ReadOnly Transactions from Mobile Clients".** Proceedings of Second IEEE Workshop on Mobile Computing Systems and Applications, 1999.

[LAM et al., 1999b] LAM, K; CHAN, E; LEUNG, H; AU, M. **"Concurrency Control Strategies for Ordered Data Broadcast in Mobile Computing Systems".** Ieice trans. inf. syst., Vol.E82-D, nº.3. Março, 1999.

[LAM et al., 2000] LAM, K; KUO, T; TSANG, W; LAW, K. **"Concurrency Control in Mobile Distributed Real-time Database Systems"**, Information Systems, vol. 25, no. 4, pp. 261-322, Junho, 2000.

[LEE & LAM, 1999] LEE, V; LAM, K. **"Optimistic Concurrency Control in Broadcast Environments: Looking Forward at the Server and Backward at the Clients**". In Proceedings of International Conference on Mobile Data Access, Lecture Note in Computer Science, vol.1748, pp.97–106, Springer, 1999.

[LEE et al., 1999] LEE, S; HWANG, C; Yu, H. **"Supporting transactional cache consistency in mobile database systems"**. Proceedings of the first ACM international workshop on Data engineering for wireless and mobile access, ACM Press, Seattle, Washington, United States, 1999.

[LEE et al., 2000] LEE, M; LEE, J; MOON, S. **"Optimistic Scheduling for Transaction Management in Mobile Database Systems"**. IEICE TRANS. INF & SYST., Vol. E82-D, nº 3. Março, 2000.

[LEE & KITSUREGAWA, 2002] LEE, S; KITSUREGAWA, M. **"Efficient Processing of Wireless Read-only Transactions in Data Broadcast"**. RIDE, 2002.

[LEE et al., 2002] LEE, V; LAM, K; SON, S; CHAN, E. **"On Transaction Processing with Partial Validation and Timestamp Ordering in Mobile Broadcast Environments "**. IEEE Transactions on Computers, Special issue on Database Management Systems and Mobile Computing. Vol. 51, nº. 10. October, 2002.

[LOH et al., 2000] LOH, Y; HARA, T; TSUKAMOTO, M; NISHIO, S. **"A hybrid method for concurrent updates on disconnected databases in mobile computing environments"**. Proceedings of the 2000 ACM symposium on Applied computing, ACM Pres, pp. 563-565, Como, Italy, 2000.

[LIN & NOLTE, 1983] LIN, W; NOLTE, J. **"Basic Timestamp, Multiple Version Timestamp, and Two-Phase Locking"**. In 9th International Conference on Very Large Data Bases, October, Proceedings VLDB, pg 109-119, November, Florence, Italy, 1983.

[LIU et al., 1995] LIU, G; MARLEVI, A; MAGUIRE, G. **"A Mobile Virtual-Distributed System Architecture for Supporting Wireless Mobile Computing and Communications".** ACM – MOBICOM, pág. 111-118, 1995.

[LIU et al., 1999] LIU, P; AMMANN, P; JAJODIA, S. **"Incorporating Transaction Semantics to Reduce Reprocessing Overhead in Replicated Mobile Data Applications".** In ICDCS'99: Proceedings 19th IEEE International Conference on Distributed Computing Systems, Austin, TX, Junho, 1999.

[LIU & MAGUIRE, 1995] LIU, G; MAGUIRE, G. **"Efficient Mobility management support for wireless data services"**. To appear at the proc. of 45th IEEE Vehicular Technology Conference, Chicago, Illinois, July 1995.

[LU & SATYANARAYA, 1996] LU, Q; SATYANARAYA, M. **"Isolation-Only transactions for Mobile Computing"**. ACM Operating Systems Review, April, 1996.

[MADRIA, 1998] MADRIA, K. **"Transaction Models for Mobile Computing"**. In proceedings of 6 th IEEE Singapore International Conference on Network, World Scientific, Singapore, July, 1998.

[MADRIA & BHARGAVA, 1998a] MADRIA, K; BHARGAVA, B; **"A Transaction Model for Mobile Computing"**. International Database Engineering and Application Symposium, 1998.

[MADRIA & BHARGAVA, 1998b] MADRIA, K; BHARGAVA, B. "**On the Correctness of a Transaction Model for Mobile Computing**". In 9th International Conference and Workshop on Database and Expert Systems (DEXA), 1998.

[MADRIA et al., 1998] MADRIA, S. K.; MOHANIA, M.; RODDICK, F.; **"A Query Model for Mobile Computing using Concept Hierarchies and Summary Databases"**. 1998.

[MATEUS & LOUREIRO, 1998] MATEUS, R; LOUREIRO, A. **"Introdução à Computação Móvel",** 1998.

[MAZUMDAR & CHRYSANTHIS, 1999] MAZUMDAR, S; CHRYSANTHIS, P. **"Achieving Consistency in Mobile Databases through Localization in PRO-MOTION"**. In Proc. DEXA Intl. Workshop on Mobility in Databases and Distributed Systems, pp. 82-89, 1999.

[MOHAN et al., 1986] MOHAN, C; LINDSAY, B; OBERMARCK, R. **"Transaction Management in the R* Distributed Database Management System"**, ACM Trans. on Database Systems, 11(4), 1986.

[NEUMANN & MASKARINEC, 1997] NEUMANN, K; MASKARINEC, M. **"Mobile Computing within a Distributed Deductive Database"**. Proceedings of the ACM, 1997.

[ÖZSU & VALDURIEZ, 1999] ÖZSU, T; VALDURIEZ, P. **"Principles of Distributed Database Systems"**. Prentice Hall. Ed. 2º, 1999.

[PHATAK et al., 1995] PHATAK, P; SHIRISH. H; BADRINATH, R. "**Bounded Locking for Optimistic Concurrency Control"**. Department of Computer Science, University Rutgers, Piscatway, 1995.

[PAVLOVA & HUNG, 1999] PAVLOVA, E; HUNG, V. **"A Formal Specification of the Concurrency Control in Real-Time Databases"**. Technical Report 152, UNU/IIST, P.O.Box 3058, Macau, January 1999.

[PITOURA, 1999] PITOURA, E. **"Scalable Invalidation-Based Processing of Queries in Broadcast Push-Based Delivery"**. Proceedings of the Mobile Data Access Workshop, in coopeation with the 17th International Conference on Conceptual Modeling (ER'98). In Advances in Database Technologies, Vol. 1, pp. 230--240, LNCS Vol. 1552, Spinger Verlag 1999.

[PITOURA & BHARGAVA, 1994a] PITOURA, E; BHARGAVA, B. **"Building Information Systems for Mobile Environments"**. ACM, 1994.

[PITOURA & BHARGAVA, 1994b] PITOURA, E; BHARGAVA, B. **"Revising Transaction Concepts for Mobile Computing".** In Proceedings of the 1st IEEE Workshop on Mobile Computing Systems and Applications (MCSA94), pages 164-169, 1994.

[PITOURA & BHARGAVA, 1995] PITOURA, E; BHARGAVA, B. **"Mantaining Consistency of Data in Mobile Distributed Environment"**. In 15^{th} Int. Conference on Distributed Computer Systems, Vancouver Canada, Maio 1995.

[PITOURA & BHARGAVA, 1999] PITOURA, E; BHARGAVA, B. **"Data Consistency in Intermittently Connected Distributed Systems"**. In transactions on Knowledge and Data Engineering, Nov 1999.

[PITOURA & CHRYSANTHIS, 1999a] PITOURA, E; CHRYSANTHIS, P. **"ExploitingVersions for Handling Updates in Broadcast Disks"**. In Proceedings of International Conference in Very Large Databases pp.114–125, 1999.

[PITOURA & CHRYSANTHIS, 1999b] PITOURA, E; CHRYSANTHIS, P. **"Scalable Processing of Read-Only Transactions in Broadcast Push"**. IEEE, International Conference on Distributed Computing Systems, pg. 432-439, Austin, 1999.

[PITOURA & CHRYSANTHIS, 2002] PITOURA, E; CHRYSANTHIS, P. **"Multiversion Data Broadcast".** In IEEE Transactions on Computers 51(10):1224-1230, October, 2002.

[PITOURA & SAMARAS, 1998] PITOURA, E; SAMARAS, G. **"Data Management for Mobile Computing"**. Technical report, MITL, Kluwer, Boston, 1998.

[PREGUIÇA et al., 2003] PREGUIÇA, N; MARTINS, J; CUNHA, M; DOMINGOS, H. "**Reservations for Conflict Avoidance in a Mobile Database System".** In the Proceedings of The First International Conference on Mobile Systems, Applications, and Services (MobiSys 2003), Usenix Association, 2003

[RAMAMRITHAM & CHRYSANTHIS, 1997] RAMAMRITHAM, K; CHRYSANTHIS, P. **"Advances in Concurrency Control and Transaction Processing"**, IEEE Computer Society Press, 1997.

[REDDY & KITSUREGAWA, 1998] REDDY, K; KITSUREGAWA, M.**"Improving Performance in Distributed Database Systems Using Speculative Transaction Processing".** In Proc. of the ACM SIGMOD,1998.

[REN & DUNHAM, 2000] REN, Q; DUNHAM, M. **"Using semantic caching to manage location dependent data in mobile computing"**. In The Sixth Annual International Conference on Mobile Computing and Networking (MobiCom'00), August, 2000.

[ROCHA, 2001] ROCHA, Ricardo C. Antunes. **"Uma Arquitetura para Simulação Flexível de Protocolos para Computação Móvel"**. Instituto de Matemática e Estatística da Universidade de São Paulo. São Paulo, maio de 2001.

[SATYANARAYANAN, 1996] SATYANARAYANAN, M. **"Fundamental Challenges in Mobile Computing"**. ACM, 1996.

[SEYDIM, 1999] SEYDIM, A. **"An Overview of Transaction models in mobile environments"**. The paper is prepared for Distributed Database Management course, November, 1999.

[SEYDIM et al., 2001] SEYDIM, A; DUNHAM, M; KUMAR, V. **"Location dependent query processing"**. ACM Press, Proceedings of the 2nd ACM international workshop on Data engineering for wireless and mobile access, pg 47-53, Santa Barbara, California, United States, 2001.

[SEYDIM & DUNHAM, 2000] SEYDIM, A; DUNHAM, M. **"A Location Dependent Benchmark with Mobility Behavior"**.

[SHANMUGASUNDARAM et al., 1999] SHANMUGASUNDARAM, J; NITHRAKASHYAP, A; SIVASANKARAN R. **"Efficient Concurrency Control for Broadcast Environments"**. In Proceedings of ACM SIGMOD International Conference on Management of Data, pp.85–96, 1999.

[SILBERSCHATZ et al., 1999] SILBERSCHATZ A., KORTH H., SUDARSHAN S, **"Sistemas de Banco de Dados"**. Ed. Makron Books - SP, 3ª edição, 1999.

[SRINIVASA et al.,2001] SRINIVASA, R; WILLIAMS, C; REYNOLDS P. F. "**Distributed Transaction Processing on an Ordering Network"**. Technical Report CS-2001-08, Department of Computer Science, University of Virginia, Feb 2001.

[TANEMBAUM, 1998] TANEMBAUM, R. **"Redes de Computadores"**. 1998.

[TEWARI & GRILLO, 1995] TEWARI, R; GRILLO, P. **"Data Management for Mobile Computing on the Internet"**. ACM, 1995.

[THOMAS, 1979] THOMAS, R. H. **"A Majority Consensus Approach Concurrency Control for Multiple Copy Databases"**. TODS 4(2): 180-209, 1979.

[XU & ,2000] XU, J; L. D. K. **"Querying location-dependent data in wireless cellular environment"**. In W3C and WAP Workshop on Position Dependent Information Services, France, February 15-16 2000.

[ZHENG et al., 2002] ZHENG, B; XU, J; LEE, D. "**Cache Invalidation and Replacement Strategies for Location-Dependent Data in Mobile Environments**". IEEE Transactions on Computers, Special issue on Database Management Systems and Mobile Computing, vol. 51, nº. 10, pg 1141-1153, Oct. 2002.

[YEO & ZASLAVSKY, 1994] YEO, H; ZASLAVSKY, A. **"Submission of transactions from Mobile Workstations in a Cooperative Multidatabase Processing Environment".** International Conference on Distributed Computing Systems. IEEE, 1994.

[WALBORN & CHRYSANTHIS, 1995] WALBORN, G; CHRYSANTHIS, K. "**Supporting Semantics-Based Transaction Processing in Mobile Database Applications"**. in Proceeding of 14th IEEE Symposium on Reliable Distributed Systems, pp.31-40, setembro, 1995.

[WALBORN & CHRYSANTHIS, 1999] WALBORN, D; CHRYSANTHIS, K. **"Transaction Processing in PRO-MOTION".** Proceeding of the 1999 ACM Symposium on Applied Computing, SAC '99, pp. 89-398, San Antonio, TX, USA, 1999.